ELECTRICAL/ELECTRONICS EQUIPMENT SERVICING

Level – III

Based on April 2022, Curriculum Version



Module Title: Assemble, enter and verify operating instructions in microprocessor equipped devices

Module Code: EIS EEES4 M03 1023

Nominal duration: 120 Hours

Prepared by: Ministry of Lobar and Skill

Ocb, 2023

Addis Ababa, Ethiopia

Table of Contents

Acknowledgement **3**
Introduction to the Module **5**
Unit one: Assemble and enter operating instructions **6**
1.1. Understand basic concept of microprocessor **7**
1.2. Applying work health and safety (OHS). **9**
1.2.1. Occupational Health and Safety 9
1.3. Obtaining and identifying tools, equipment and testing instruments **10**
1.4. Consulting supervisor/customers. **12**
1.4.1. Consultation processes 12
1.4.2. The process consulting mode 13
1.5. Checking device installation for compliance **15**
Self Check 1.1. ..... **17**
Unit Two: Write assembly language instructions **18**
2.1. Introducing machine languages. **19**
2.2. Preparing basic flow chart for the intended assembly language **20**
2.2.1 Basic Flow Chart 20
2.3. Preparing machine language in assembly language **21**
2.3.1. Microcontrollers, Assembly Language, and Embedded Systems 25
2.3.2. The Building Blocks of an Embedded System 25
2.4. Understanding logic instructions for registers. **29**
2.4.1. Instructions of Assembly Language 31
2.4.2. Logical Instructions 32
2.4.3. Assembly Conditions 36
2.4.4. Write an assembly language program 39
2.4.5. Development Tools For Assembly Language Program 39
2.4.6. Interfacing Input and output (I/O) devices 42
2.4.7. Assembly language **43**
1.5. Preparing assembly language **48**
1.6. Implementing basic software technique **50**
1.6.1. Detail command based on the logic instructions. 51
Self Check 2.1. ..... **54**
Unit Three: Assemble and enter operating instructions **55**
3.1. Checking and isolating circuits/machines/plant. **56**
3.1.1. Understanding Microprocessor Equipment 56
3.2. Identifying microprocessors equipped devices **58**
3.2.1. Operating instructions for a microprocessor application 60
3.3. Checking entered data. **62**
Self Check 3.1. ..... **63**
Unit Four : Test device operation and report **64**
4.1. Testing device operation using tools, materials and test instruments **65**
4.2. Identifying and correcting operating anomaly **69**
4.4. Documenting work completion **72**
4.5. Disposing waste materials. **73**
Self Check 4.1. ..... **75**

Acknowledgement

Ministry of Labor and Skills wish to extend thanks and appreciation to the many representatives of TVET instructors and respective industry experts who donated their time and expertise to the development of this Teaching, Training and Learning Materials (TTLM).

Acronyms

ALU	arithmetic and logic unit
TTL	Transistor-Transistor Logic
ECL	Emitter Coupled Logic
CMOS	Complementary Metal-Oxide Semiconductor
DVD	Digital Versatile Disc
ICs	Integrated Circuits
AC	Alternating current
DC	Direct current
OHS	Occupational health and safety
LAP	Learning Assistance Program
LED	light emitting diode
PCB	printed circuit boar
ESD	Electro-Static Discharge
DMM	Digital Muli-Meter
VOM	Volt Ohm Meter

Introduction to the Module

In Electrical/electronic equipment servicing filed; the: **Assemble, enter and verify operating instructions in microprocessor equipped devices** project helps to require **Assemble, enter and verify operating instructions in microprocessor equipped devices** includes constructing, diagnosing faults, dismantling, re-assembling, testing and preparing report.

This module is designed to meet the industry requirement under the Electrical/electronic equipment servicing occupational standard, particularly for the unit of competency: **Assemble, enter and verify operating instructions in microprocessor equipped devices Module units**

- **Prepare to assemble and enter operating instructions**
- **Write assembly language instructions**
- **Assemble and enter operating instructions**
- **Test device operation and report**

Learning objectives of the Module

At the end of this session, the students will able to:

- **Prepare to assemble and enter operating instructions**
- **Write assembly language instructions**
- **Assemble and enter operating instructions**
- **Test device operation and report**

Module Learning Instructions:

1. Read the specific objectives of this Learning Guide.
2. Follow the instructions described below.
3. Read the information written in the information Sheets
4. Accomplish the Self-checks
5. Perform Operation Sheets
6. Do the “LAP test”

Unit one: Assemble and enter operating instructions

This learning unit is developed to provide the trainees the necessary information regarding the following content coverage and topics:

Assemble and enter operating instructions

- Introducing basic concept of microprocessor.
- Applying work health and safety (OHS).
- Obtaining and identifying tools, equipment and testing instruments
- Consulting supervisor/customers.
- Checking device installation for compliance

This unit will also assist you to attain the learning outcomes stated in the cover page. Specifically, upon completion of this learning guide, you will be able to:

- Understand basic concept of microprocessor.
- Apply work health and safety (OHS).
- Obtain and identify tools, equipment and testing instruments
- Consult supervisor/customers.
- Check device installation for compliance

1.1. Understand basic concept of microprocessor

A microprocessor is a programmable electronics chip that has computing and decision making capabilities similar to central processing unit of a computer. Any microprocessor based systems having limited number of resources are called microcomputers. Nowadays, microprocessor can be seen in almost all types of electronics devices like mobile phones, printers, washing machines etc. Microprocessors are also used in advanced applications like radars, satellites and flights. Due to the rapid advancements in electronic industry and large scale integration of devices results in a significant cost reduction and increase application of microprocessors and their derivatives.

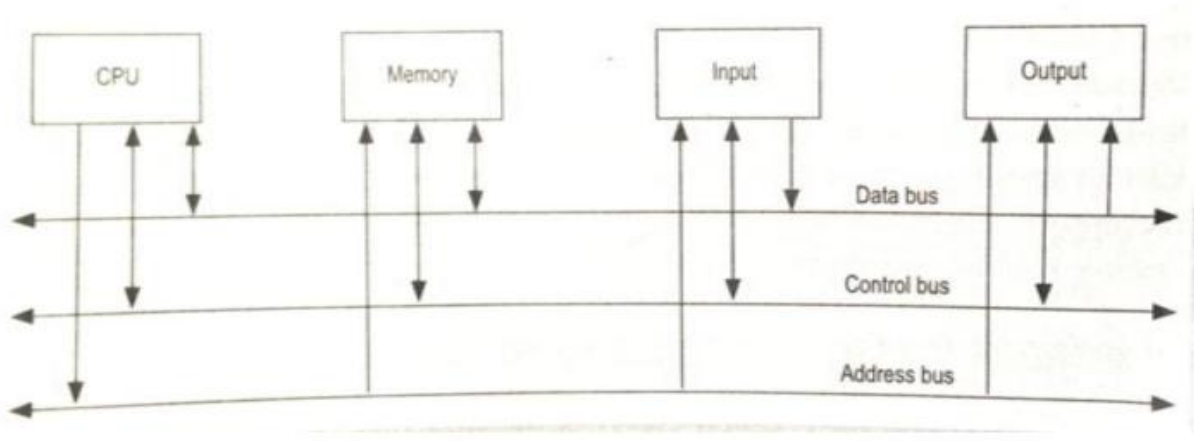


Fig.1 Microprocessor-based system

- **Bit:** A bit is a single binary digit.
- **Word:** A word refers to the basic data size or bit size that can be processed by the Arithmetic and logic unit of the processor**.** A 16-bit binary number is called a word in a 16-bit processor.

Bus: A bus is a group of wires/lines that carry similar information.

- **System Bus:** The system bus is a group of wires/lines used for communication between the microprocessor and peripherals.
- **Memory Word:** The number of bits that can be stored in a register or memory element is called a memory word.
- **Address Bus:** It carries the address, which is a unique binary pattern used to identify a memory location or an I/O port. For example, an eight bit address bus has eight lines and thus it can address 28 = 256 different locations. The locations in hexadecimal format can be written as 00H – FFH.

- **Data Bus:** The data bus is used to transfer data between memory and processor or between I/O device and processor. For example, an 8-bit processor will generally have an 8-bit data bus and a 16-bit processor will have 16-bit data bus.

Control Bus: The control bus carry control signals, which consists of signals for selection of memory or I/O device from the given address, direction of data transfer and synchronization of data transfer in case of slow devices. A typical microprocessor consists of arithmetic and logic unit (ALU) in association with control unit to process the instruction execution. Almost all the microprocessors are based on the principle of store-program concept. In store-program concept, programs or instructions are sequentially stored in the memory locations that are to be executed. To do any task using a microprocessor, it is to be programmed by the user. So the programmer must have idea about its internal resources, features and supported instructions. Each microprocessor has a set of instructions, a list which is provided by the microprocessor manufacturer. The instruction set of a microprocessor is provided in two forms: binary machine code and mnemonics. Microprocessor communicates and operates in binary numbers 0 and 1. The set of instructions in the form of binary patterns is called a machine language and it is difficult for us to understand. Therefore, the binary patterns are given abbreviated names, called mnemonics, which forms the assembly language. The conversion of assembly-level language into binary machine-level language is done by using an application called assembler. Technology Used:

The semiconductor manufacturing technologies used for chips are:

- Transistor-Transistor Logic (TTL)
- Emitter Coupled Logic (ECL)
- Complementary Metal-Oxide Semiconductor (CMOS)

Classification of Microprocessors: Based on their specification, application and architecture microprocessors are classified. Based on size of data bus:

- 4-bit microprocessor
- 8-bit microprocessor
- 16-bit microprocessor
- 32-bit microprocessor

• **Based on application:** General-purpose microprocessor- used in general computer system and can be used by programmer for any application. Examples, 8085 to Intel Pentium. Microcontroller- microprocessor with built-in memory and ports and can be programmed for any generic control application. Example, 8051. Special-purpose processors- designed to handle special functions required for an application. Examples, digital signal processors and application-specific integrated circuit (ASIC) chips. Based on architecture:

Reduced Instruction Set Computer (RISC) processors

- Complex Instruction Set Computer (CISC) processors

• 2. 8085 MICROPROCESSOR ARCHITECTURE The 8085 microprocessor is an 8-bit processor available as a 40-pin IC package and uses +5 V for power. It can run at a maximum frequency of 3 MHz. Its data bus width is 8-bit and address bus width is 16-bit, thus it can address 216 = 64 KB of memory. The internal architecture of 8085

The microprocessor is a continuation of the trend toward smaller computers that began in the middle 1960s. In the early years of computer development the ea4>hasis was on larger and more powerful machines. Computers were so expensive that only large institutions could own them and only specially trained personnel could operate them. New technologies, such as transistors and integrated circuits, provided greater speed but did not reduce costs. Computers remained remote and mysterious objects. Nfinicomputers started the present trend. The first minicomputers were primitive by computer standards and still cost tens of thousands of dollars. Nevertheless, laboratories, factories, and smaller institutions that could not afford large com puters could now purchase a small computer like the Digital Equipment PDP-8, Data General Nova, Scientific Data Systems 92, or IBM 1130. Cheaper electronic dfcults led to cheaper minicomputers; by 1970 a small minicomputer for use in a laboratory, office, factory, warehouse, or classroom cost a few thousand dollars. Yet developments in integrated circuits went far beyond reducing the cost of minicomputers. Soon the construction of a single integrated circuit that could perform the functions of a computer became possible.

1.2. Applying work health and safety (OHS).

1.2.1. Occupational Health and Safety

Occupational health and safety is the field of public health that studies trends in illnesses and injuries in the worker population and proposes and implements strategies and regulations to prevent them. Its scope is broad, encompassing a wide variety of discipline from toxicology and epidemiology to ergonomics and violence prevention.

Workplace safety is an important part of any job and requires that everyone in the company adhere to the safety guidelines and policies in place. Carefully following appropriate safety guidelines can go a long way toward preventing workplace injuries. Here are some ways you can work to stay safe on the job**.**

A. Core OHS principles

Occupational health and safety is an extensive multidisciplinary field, invariably touching on issues related to scientific areas such as medicine

- ✓ including physiology and toxicology
- ✓ Ergonomics, physics and chemistry, as well as technology, economics, law and other areas specific to various industries and activities. Despite this variety of concerns and interests, certain basic principles can be identified, including the following.

• **All workers have rights**. Workers, as well as employers and governments, must ensure that these rights are protected and must strive to establish and maintain decent working conditions and a decent working environment. More specifically:

- ✓ Work should take place in a safe and healthy working environment.
- ✓ Conditions of work should be consistent with workers' well-being and human dignity;
- ✓ Work should offer real possibilities for personal achievement, self fulfillment and service to society

• **Occupational health and safety policies must be established**. Such policies must be implemented at both the national (governmental) and enterprise levels. They must be effectively communicated to all parties concerned.

• **A national system for occupational safety and health must be established**. Such a system must include all the mechanisms and elements necessary to build and maintain a preventive safety and health culture. The national system must be maintained, progressively developed and periodically reviewed.

• **A national programme on occupational safety and health must be formulated**. Once formulated, it must be implemented, monitored, evaluated and periodically reviewed.

• Social partners (that is, employers and workers) and other stakeholders must be consulted. This should be done during formulation, implementation and review of all policies, systems and programmer.

• **Occupational safety and health programmers and policies must aim at both prevention and protection.** Efforts must be focused above all on primary prevention at the workplace level. Workplaces and working environments should be planned and designed to be safe and healthy.

• **Continuous improvement of occupational safety and health must be promoted**. This is necessary to ensure that national laws, regulations and technical standards to prevent occupational injuries, diseases and deaths are adapted periodically to social, technical and scientific progress and other changes in the world of work.

1.3. Obtaining and identifying tools, equipment and testing instruments

When it comes to assembling microprocessors, there are several tools, equipment, and testing instruments that are commonly used. These tools and instruments help in the efficient and accurate assembly of

microprocessors, ensuring their proper functioning. In this comprehensive response, we will discuss various tools, equipment, and testing instruments used in microprocessor assembly

A. Soldering Irons and Stations:

Soldering irons and stations are essential tools for microprocessor assembly. They are used to solder the various components onto the microprocessor board. Soldering irons come in different wattages and tip sizes to accommodate different soldering requirements. Soldering stations provide a controlled temperature environment for precise soldering. They often include features such as adjustable temperature control, digital displays, and safety features like grounding.

B. Tweezers and ESD

(Electrostatic Discharge)-safe tools are crucial for handling delicate microprocessor components. Tweezers with fine tips allow for precise placement of small components onto the microprocessor board. ESD-safe tools are designed to prevent electrostatic discharge from damaging sensitive electronic components. These tools are made from materials that dissipate static electricity safely.

C. Desordering Tools

Desoldering tools are used to remove or replace faulty components on the microprocessor board. These tools include disordering pumps, desoldering wicks, and desoldering stations. Desoldering pumps use suction to remove molten solder from joints, while desoldering wicks absorb molten solder through capillary action. Desoldering stations provide a controlled environment for efficient component removal.

D. Hot air rework stations

Hot air rework stations are used for tasks such as removing surface-mounted components or reflowing solder on the microprocessor board. These stations consist of a heating element and a nozzle that directs hot air onto the targeted area. The controlled airflow and temperature allow for precise heating and removal of components without damaging the surrounding circuitry.

E. A microscope or magnifying lamp

A microscope or magnifying lamp is essential for inspecting and working on small-scale components during microprocessor assembly. These tools provide magnification, allowing technicians to examine intricate details and ensure accurate placement of components. They are particularly useful for tasks that require high precision, such as soldering tiny connections or inspecting solder joints.

F. ESD protection equipment

ESD protection equipment is crucial in microprocessor assembly to prevent electrostatic discharge from damaging sensitive components. This includes ESD mats, wrist straps, and grounding cords. ESD mats provide a static-dissipative work surface, while wrist straps and grounding cords ensure that technicians are grounded, preventing the buildup of static electricity.

G. Testing instruments

Various testing instruments are used to verify the functionality and performance of assembled **microprocessors. These instruments include:**

H. Multi-meters: Multi-meters are versatile instruments used to measure voltage, current, resistance, and continuity. They are commonly used to troubleshoot and test electrical connections on microprocessor boards.

I. Logic Analyzers: Logic analyzers capture and analyze digital signals in real-time. They help in debugging complex digital circuits by providing insights into signal timing, protocol analysis, and system behavior.

J. Oscilloscopes: Oscilloscopes are used to visualize and measure electrical waveforms. They enable technicians to analyze the timing and shape of signals in microprocessor circuits, helping identify issues such as noise, glitches, or improper signal levels.

In conclusion, assembling microprocessors requires a range of tools, equipment, and testing instruments to ensure accurate placement of components and proper functionality of the final product. Soldering irons, tweezers, desoldering tools, hot air rework stations, microscopes or magnifying lamps, ESD protection equipment, and various testing instruments are essential for successful microprocessor assembly.

1.4. Consulting supervisor/customers.

1.4.1. Consultation processes

The basic concept of process consultation is defined and compared to other major consultation concepts. Process consultation is a philosophy about and attitude toward the process of helping individuals, groups, organizations, and communities. It is not merely a set of techniques to be compared to and contrasted with other techniques. Process consultation is the key philosophical underpinning to organizational learning and development in that most of what the consultant does in helping organizations is based on the central assumption that one can only help a human system to help itself. The consultant never knows enough about

the particular situation and culture of an organization to be able to make specific recommendations on what the members of that organization should do to solve their problems.

Process Consultation is the creation of a relationship with the client that permits the client to perceive, understand, and act on the process events that occur in the client's internal and external environment in order to improve the situation as defined by the client.

An organizational consultant is working with managers to improve some aspects of the organization; the same fundamental dynamics are involved. What goes on between a helper and the person or group being helped is what I have called process consultation or PC for short.

A. Modes of consulting

Consulting is characterized by different approaches which reflect fundamental assumptions about the consultant's role. All managers have their own approach to the tasks they face and the way they deal with people. This determines the manager's style. A critical element here is the perception the manager has about their fundamental role in the organization. These are referred to as modes. Process Consultation, characterized three basic modes based on the relationship between the consultant and the client: the expert, the doctor–patient and the process consulting modes.

The expert mode

In the expert mode the client identifies a particular problem with the business, analyses the problem and articulates it to the consultant. The consultant then uses their expertise to identify a solution to the problem. This form of consulting is often found in areas where the consultant has a specialist knowledge which the client organization recognizes that it lacks.

The doctor–patient mode

The doctor–patient mode is also characterized by the consultant acting as an expert. In this mode, however, the consultant also takes responsibility for diagnosing the problem in the first place. The client may just express an opinion that the business 'could be better' in some way or that 'something is not quite right'. Again, the consultant is expected to contribute specialist knowledge and insights to the business.

1.4.2. The process consulting mode

Both the expert mode and the doctor–patient mode demand that the consultant, an outsider, offers a well-considered, expert solution – a prescription – to address the problems that the business faces. Process consulting takes a different stance. It is based on the premise that the only people who can help the business are the people who make it up. The consultant, as an outsider, cannot impose a solution on the organization.

What the consultant can do is assist those who make up the organization to recognize problems and then discover the solutions to them. The consultant is not so much an expert, more a facilitator of change.

The process of consultation is an extremely important concept in the context of managing an organization. Organizations exist to create value for stakeholders and consultation is a process by which the management of the organization aims to better understand the needs, wants and expectations of stakeholders, so that value can be created. Consultation is an active process in which organization management opens formal and informal communication channels between the organization and its stakeholders.

- **These formal and informal communication channels might include:**
  - ✓ Open meetings e.g. stakeholders are invited to come to an open meeting or a series of meetings
  - ✓ Surveys e.g. stakeholders are invited to complete a survey (paper or online type)
  - ✓ Focus group e.g. a select cross-section of stakeholders, small in number, are invited to attend a meeting or series of meetings
  - ✓ Invitation to send a written response e.g. stakeholders are invited to submit comments in writing on a proposal or plan
  - ✓ Informal meetings e.g. organization management might mingle with people at an event a canvass certain ideas and see what response they get

B. The purpose of consultation

A. To invite stakeholders to provide advice to the management of the organization about their needs, wants and expectations. In other words, tell the organization what value it wants and how it can provide this value.

B. To invite stakeholders to comment on plans those have been created by organization management to provide this value requested by stakeholders.

C. To quell any criticism that organization management have not taken account of, or are not listening to the needs of stakeholders in developing strategic and operational plans.

There is a widespread view that if a plan is conceived without proper consultation with stakeholders then it has far less chance of successful implementation. There is a clear need for anyone responsible for the formulation of a plan to consult with all persons who will be affected by the plan. For example, a budget for any area of organization operation should not be set without consultation with people who work in that area of operation. Likewise, management should not construct a plan for a new sport program without consultation with people who likely be program users. Setting an Operational Plan without consultation disadvantages the organization because:

- ✓ A lack of consultation fails to take advantage of all available knowledge and expertise

- ✓ A lack of consultation makes people feel left out and creates negativity toward the emerging plan.

1.5. Checking device installation for compliance

To check device installation for compliance of microprocessors, there are several steps and considerations that need to be taken into account. This process ensures that the microprocessors are installed correctly and meet the necessary requirements for optimal performance and functionality. In this response, we will outline the key aspects to consider when checking device installation for compliance of microprocessors.

1. **Physical Installation**: The first step in checking device installation is to ensure that the physical installation of the microprocessor is done correctly. This involves properly aligning the processor with the socket on the motherboard and securing it in place using the appropriate locking mechanism. It is crucial to follow the manufacturer's guidelines and instructions during this process to avoid any damage or improper installation.

2. **Power Supply**: Microprocessors require a stable and adequate power supply to function properly. It is essential to verify that the power supply unit (PSU) is capable of delivering sufficient power to the microprocessor. This can be done by checking the PSU's specifications and ensuring that it meets or exceeds the recommended power requirements of the microprocessor. Additionally, proper cable connections between the PSU and motherboard should be verified to ensure a reliable power supply.

3. **Cooling System**: Microprocessors generate heat during operation, and it is crucial to have an efficient cooling system in place to prevent overheating. The cooling system typically consists of a heatsink, fan, or liquid cooling solution. When checking device installation for compliance, it is important to ensure that the cooling system is properly installed and functioning correctly. This includes verifying that the heatsink or cooling solution is securely attached to the microprocessor, thermal paste or pads are applied correctly, and fans are spinning at appropriate speeds.

4. **BIOS/UEFI Settings**: The Basic Input/Output System (BIOS) or Unified Extensible Firmware Interface (UEFI) settings play a significant role in ensuring proper device installation compliance for microprocessors. These settings control various aspects of the microprocessor's operation, such as clock speed, voltage, and power management. It is important to access the BIOS/UEFI settings and verify that they are configured

correctly for the specific microprocessor being installed. This may involve enabling or disabling certain features or adjusting settings based on the manufacturer's recommendations.

5. **Operating System Compatibility**: Another aspect to consider when checking device installation compliance is the compatibility of the microprocessor with the operating system (OS) being used. Different microprocessors may have specific requirements or limitations when it comes to OS compatibility. It is essential to consult the documentation provided by the microprocessor manufacturer to ensure that the installed OS is supported and that any necessary drivers or updates are installed.

6. **Testing and Benchmarking**: Once the device installation is complete, it is recommended to perform testing and benchmarking to ensure that the microprocessor is functioning correctly and meeting performance expectations. Various software tools and utilities are available for stress testing, temperature monitoring, and benchmarking purposes. These tests can help identify any potential issues or bottlenecks in the system and allow for adjustments or optimizations if needed.

In conclusion, checking device installation for compliance of microprocessors involves several crucial steps such as physical installation, power supply verification, cooling system assessment, BIOS/UEFI settings configuration, operating system compatibility check, and testing/benchmarking. By following these steps and considering the specific requirements provided by the microprocessor manufacturer, one can ensure a compliant and well-functioning microprocessor installation.

Self Check 1.1.

TEST I: Say True or False

1. A microprocessor is a programmable electronics chip that has computing and decision making capabilities
2. ESD protection equipment is crucial in microprocessor assembly to prevent electrostatic discharge.
3. The process of consultation is an extremely important concept in the context of managing an organization.

TEST II : Chose the best answer

1._____ is the tools which used to remove or replace faulty components on the microprocessor board

A. Soldering Iron B. Desoldering Iron C. Logic Analyzers D. None

2._____ Logic analyzers capture and analyze digital signals in real-time

A. Oscilloscopes B. Desoldering Iron C. Logic Analyzers D. None

3. _____ Oscilloscopes are used to visualize and measure electrical waveforms

A. Logic Analyzers B. Function generator C. Oscilloscopes D. None

Unit Two: Write assembly language instructions

This learning unit is developed to provide the trainees the necessary information regarding the following content coverage and topics:

- Introducing to machine languages.
- Preparing basic flow chart for the intended assembly language.
- Preparing machine language in assembly language
- Identifying logic instructions for registers
- Preparing assembly language
- Implementing basic software techniques

This unit will also assist you to attain the learning outcomes stated in the cover page. Specifically, upon completion of this learning guide, you will be able to:

- Introducing machine languages.
- Prepare basic flow chart for the intended assembly language.
- Understand machine language in assembly language
- Identify logic instructions for registers, stacks operations, and interfacing devices
- Organize assembly language
- Implement basic software techniques

2.1. Introducing machine languages.

Machine languages are low-level programming languages that are directly understood and executed by a computer's hardware. They consist of a set of instructions that can be executed by the central processing unit (CPU) of a computer. In the context of instruction micro, machine languages play a crucial role in defining the operations and behavior of microprocessors.

Instruction micro refers to the design and implementation of microprocessors, which are integrated circuits that serve as the central processing unit (CPU) of a computer system. These microprocessors execute instructions stored in memory, performing arithmetic, logical, control, and input/output operations. When introducing machine languages for instruction micro, it is essential to consider the specific architecture and design principles of the microprocessor being used. Different microprocessors may have different instruction sets and formats, requiring specific machine language instructions to perform various operations.

Machine languages for instruction micro typically consist of binary code, where each instruction is represented by a sequence of 0s and 1s. Each binary instruction corresponds to a specific operation or command that the microprocessor can execute. These instructions are designed to be easily interpretable by the microprocessor's hardware.

To facilitate programming in machine language, mnemonic codes are often used as human-readable representations of machine instructions. These mnemonics provide a more intuitive way for programmers to write code in machine language. For example, instead of writing a binary sequence like "10110010," a programmer can use a mnemonic like "ADD" to represent an addition operation.

The mapping between mnemonics and their corresponding binary instructions is defined by an assembly language. Assembly languages are symbolic representations of machine languages that provide mnemonics for each machine instruction. They serve as a bridge between high-level programming languages and machine languages, making it easier for programmers to write code at a lower level of abstraction.

Assembly languages allow programmers to write code using mnemonic instructions that closely resemble the operations they want the microprocessor to perform. These mnemonic instructions are then translated into their corresponding machine language instructions by an assembler. The assembler converts the assembly code into binary machine code that can be executed by the microprocessor. In addition to mnemonics, assembly languages also support symbolic labels and directives. Labels are used to mark specific locations in the program, allowing for easier referencing and branching. Directives provide additional instructions to the assembler, such as defining constants or allocating memory.

It is important to note that different microprocessors may have different assembly languages and instruction sets. Each microprocessors architecture has its own unique set of instructions and addressing modes, which determine how operands are specified for each instruction. Therefore, when introducing machine languages for instruction micro, it is crucial to consider the specific microprocessor architecture being used.

In summary, machine languages play a vital role in instruction micro by providing a low-level programming interface for microprocessors. These languages consist of binary instructions that directly correspond to operations executed by the hardware. Assembly languages serve as a human-readable representation of machine languages, allowing programmers to write code using mnemonic instructions. Assemblers then translate this assembly code into binary machine code that can be executed by the microprocessor.

2.2. Preparing basic flow chart for the intended assembly language

The purpose of this section is to provide a basic flow chart for the intended assembly language programming for the project. The flow chart will outline the steps involved in the program's execution, from start to finish.

2.2.1 Basic Flow Chart

The following is a basic flow chart for the intended assembly language programming:

1. **Start**: The program begins at the starting address specified in the boot loader.
2. **Reset Vector**: The program jumps to the reset vector, which sets up the stack pointer and initializes the CPU registers.
3. **Main Loop**: The program enters the main loop, where it performs the desired tasks.
4. **Task 1**: The program performs task 1, such as reading input from the user or performing calculations.

5. **Task 2**: The program performs task 2, such as writing output to the screen or storing data in memory.
6. **Task 3**: The program performs task 3, such as accessing external devices or interacting with other programs.
7. **Exit**: The program exits the main loop and returns control to the operating system.

Additional Information

The following are some additional details about the assembly language programming for the project: The program will be written in 32-bit assembly language, using the x86-64 instruction set architecture. The program will use the NASM (Net wide Assembler) assembler to convert the assembly code into machine code.

The program will use the Linux operating system, and will be compiled using the GCC (GNU Compiler Collection) compiler.

The program will use the standard C library, which provides a set of functions for performing common tasks such as input/output operations and memory management.

2.3. Preparing machine language in assembly language

A. Assembly Language Programming

Assembly language unlocks the secret of your computer's hardware and software. It teaches you about the way the computer's hardware and operating system work together and how, the application programs communicate with the operating system. Assembly language, unlike high level languages, is machine dependent. Each microprocessor has its own set of instructions, that it can support. Here we will discuss, only the IBM-PC assembly language. It consists of the Intel 8086/8088 instruction set. The instructions for the Intel 8088 may be used without modification on all its enhancements - 80186,80286,80386,80486 and Pentium.

- **Assembly Language Syntax and Program Structure**

A processor can directly execute a machine language program. Though it is possible to program directly in machine language, assembly language uses mnemonics to make programming easier. An assembly language program uses mnemonics to represent symbolic instructions and the raw data that represent variables and constants. A machine language program consists of: a list of numbers representing the bytes of machine instructions to be executed and data constants to be used by the program.

- **Assembly Language Syntax**

An assembly language program consists of statements. The syntax of an assembly language program

statement obeys the following rules:

- ✓ Only one statement is written per line.
- ✓ Each statement is either an instruction or an assembler directive.
- ✓ Each instruction has an opcode and possibly one or more operands.
- ✓ An opcode is known as a mnemonic.
- ✓ Each mnemonic represents a single machine instruction.
- ✓ Operands provide the data to work with.

- **Assembler Directives**

Pseudo instructions or assembler directives are instructions that are directed to the assembler. Assembler directives affect the generated machine code, but are not translated directly into machine code. Directives can be used to declare variables, constants, segments, macros, and procedures as well as supporting conditional assembly.

In general, a directive contains pseudo-operation code, tells the assembler to do a specific thing, and is not translated into machine code.

- **Segment directives**

Segments are declared using directives. The following directives are used to specify the following segments:

- ✓ stack
- ✓ data
- ✓ code

- **Stack Segment**
  - ✓ Used to set aside storage for the stack.
  - ✓ Stack addresses are computed as offsets into this segment.
  - ✓ Use: .stack followed by a value that indicates the size of the stack.
- **Data Segment**
  - ✓ Used to set aside storage for variables.
  - ✓ Constants are defined within this segment in the program source.
  - ✓ Variable addresses are computed as offsets from the start of this segment.
  - ✓ Use: .data followed by declarations of variables or definitions of constants.
- **Code Segment**

The code segment contains executable instructions macros and calls to procedures. Use: .code followed by a sequence of program statements.

- **Memory Models**

The memory model specifies the memory size assigned to each of the different parts or segments of a program. There exist different memory models for the 8086 processor.

The 8086 microprocessor is a 16-bit microprocessor that was introduced by Intel in 1978. It is part of the x 86 families of processors and is widely used in various applications, including personal computers, embedded systems, and industrial automation.

Assembly language is a low-level programming language that is specific to particular processor architecture. It provides a direct correspondence between the instructions executed by the processor and the operations performed by the hardware. The 8086 assembly language is designed to work with the 8086 microprocessor and its successors, such as the 80186, 80286, and later generations.

Assembly language programming for the 8086 microprocessor involves writing code using mnemonic instructions that represent specific machine operations. These instructions are then assembled into machine code, which can be directly executed by the processor. Assembly language programs are typically written using a text editor and then assembled using an assembler.

The 8086 assembly language supports a wide range of instructions for performing arithmetic and logical operations, data movement, control flow, and input/output operations. These instructions can operate on various data types, including integers, floating-point numbers, and strings.

Commonly used instructions in 8086 assembly language programming include:

1. **Data Movement Instructions:** These instructions are used to move data between registers, memory locations, and immediate values. Examples include MOV (move), PUSH (push onto stack), POP (pop from stack), XCHG (exchange), LEA (load effective address), etc.

2. **Arithmetic and Logical Instructions:** These instructions perform arithmetic and logical operations on data. They include ADD (addition), SUB (subtraction), MUL (multiplication), DIV (division), AND (bitwise AND), OR (bitwise OR), XOR (bitwise XOR), etc.

3. **Control Flow Instructions:** These instructions control the flow of execution in a program. They include JMP (jump unconditional), JZ (jump if zero), JNZ (jump if not zero), JC (jump if carry), JNC (jump if not carry), CALL (call a subroutine), RET (return from subroutine), etc.

4**. String Instructions:** The 8086 microprocessor also provides a set of instructions specifically designed for

working with strings of data. These instructions include MOVSB (move byte from string to string), MOVSW (move word from string to string), CMPSB (compare byte strings), SCASB (scan byte string), etc.

Assembly language programming for the 8086 microprocessor requires a good understanding of the processor's architecture, instruction set, and memory organization. It also requires careful management of registers and memory locations to ensure efficient and correct execution of the program.

In addition to the basic instructions, the 8086 microprocessor also supports various addressing modes, which allow programmers to access data in different ways. These addressing modes include immediate addressing, register addressing, direct addressing, indirect addressing, indexed addressing, based addressing, and relative addressing.

Assembly language programming for the 8086 microprocessor can be challenging due to its low-level nature and the need for manual memory management. However, it offers a high level of control over the hardware and can result in highly optimized code.

1. "The 8086 Microprocessor: Programming and Interfacing the PC" by Kenneth Ayala - This book provides a comprehensive guide to programming the 8086 microprocessor using assembly language. It covers topics such as instruction set architecture, memory organization, I/O operations, and interrupts handling.

2. **Intel 8086/8088 Assembly Language Programming Manual -** This manual published by Intel Corporation provides detailed information about assembly language programming for the 8086 microprocessor. It includes a complete instruction set reference, examples, and programming techniques.

3. "Assembly Language for x86 Processors" by Kip R. Irvine - This book is a widely used textbook for learning assembly language programming for x86 processors, including the 8086. It covers topics such as data representation, memory addressing, arithmetic and logical operations, control flow, and string manipulation.

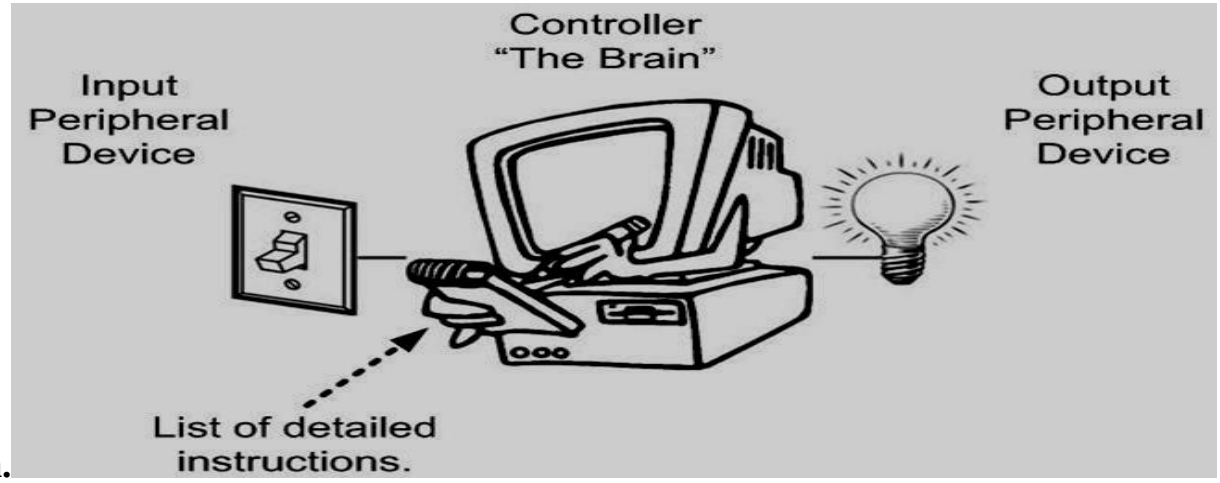
These authoritative references provide in-depth information and guidance on assembly language programming for the 8086 microprocessor, making them valuable resources for both beginners and experienced programmers in this domain.

2.3.1. Microcontrollers, Assembly Language, and Embedded Systems

A. WHAT IS AN EMBEDDED SYSTEM

• An embedded system is an electronic system that contains at least one controlling device, i.e. “the brain”, but in such a way that it is hidden from the end user. That is, the controller is embedded so far in the system that usually users don’t realize its presence..

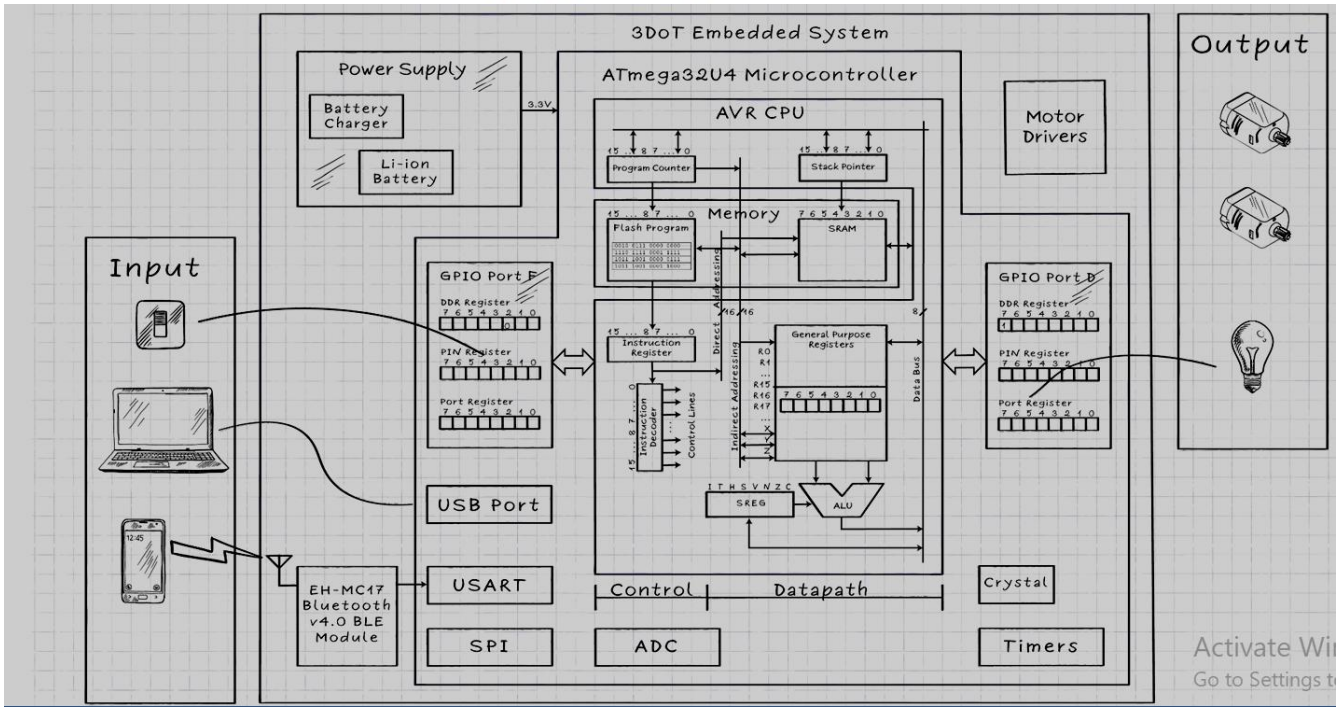
Embedded systems perform a dedicated function.



EE	Technology	Tools
EE201	Discrete Logic	Boolean Algebra
EE301	Field Programmable Gate Array (FPGA), Application-Specific Integrated Circuit	HDL (typically VHDL or Verilog)
EE346	**Microcontroller**	Program (typically C++ or
EE443	System on a Chip (SoC)	System Level Design Language

2.3.2. The Building Blocks of an Embedded System

“This course will be an introduction to modern **RISC** based **microcontrollers** and assembly language programming. We will use the **Atmel AVR** family of microcontrollers to teach hardware design of small, minimum-component systems performing simple task-oriented activities.” Source: EE346 Syllabus



- WHAT IS AN ARDUINO.

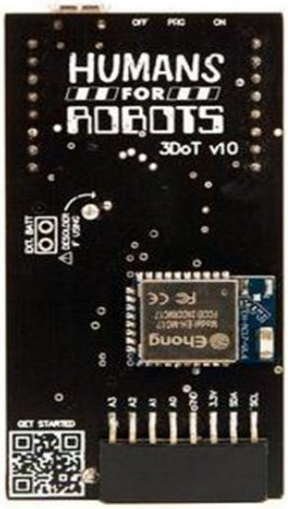


- Arduino is an open-source electronics PCB containing a microcontroller and the things needed to support it: Power Supply Communications, Reset Button, Clock, and Connectors for adding Sensors and Actuators in the physical world.
- Using an Arduino you can develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs.
- The Arduino consists of two parts; the hardware and the software.
- Our Robot Board is based on the Arduino Leonardo which contains an **ATmega32U4** 8 bit microcontroller. We will be using AVR Studio to develop the software for the

3DoT (The **3D o**f **T**hings) is a micro-footprint 3.5 x 7 cm all-in-one Arduino compatible microcontroller board designed for robot projects by Humans for Robots.

- **Microcontroller**: ATmega32U4
- **Bluetooth**: FCC-certified BLE 5.0 module
- **Power Management**:
    - RCR123A battery holder
    - Included 600 mAh rechargeable battery
    - Microchip MCP7383 battery charge controller
    - External battery connector – for input voltages between 4 – 18 V
    - Reverse polarity protection – plug in the battery backwards? No problem





- **Motors & Servos**:
    - ✓ 2x JST motor connectors
    - ✓ 2x standard servo connectors
- **Expansion**:

✓ 16-pin top female headers for shields – providing I/O, I²C, SPI, USART, 3.3 V and 5 V.

✓ Forward-facing 8-pin female header for sensor shields – providing 4 analog pins, I²C, and 3.3 V power

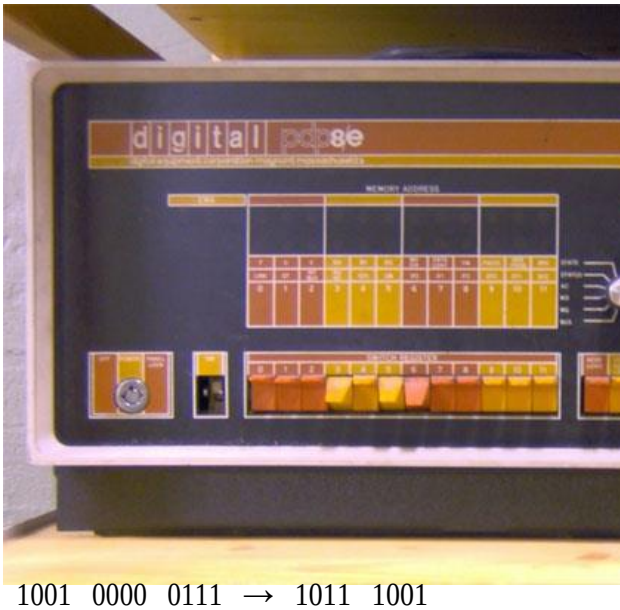
✓ For sensor shields like infrared or metal-detecting shields. Great location for headlights,

lasers, ultrasonics,

- **Programming switch**: Three-position switch for easy programming
  - ✓ No more double-tapping a button and rushing to program your board, or your robot trying to drive away while programming. Set the switch to PRG to program, RUN to execute your code.
  - ✓ Arduino in place of the Arduino IDE and associated Scripting Languag

WHAT IS A PROGRAM?

- The Program is a "very specific list of instructions" to the computer.
- The process of "creating the program" is where much of an electrical engineer's time is spent.
- The program is often referred to as Software, while the physical system components are called Hardware. Software held within non-volatile memory is called Firmware.



- Software design is all about creating patterns of 0's and 1's in order to get the computer to do what we want. These 0's and 1's are known as Machine Code.

0010 0111 0000 0000 → 1110 1111 0001 1111 → 1011 1001 0000 0111 → 1011 1001 0001 1000

1011 1001 0000 0100 → 1011 0000 0111 0110 → 1011 1000 0111 0101 → 1100 1111 1111 1101

- The architecture of the processer (or computer) within a microcontroller is unique as are the Machine Code Instructions it understands.

0010 0111 0000 0000

1110 1111 0001 1111

- The list of Machine Code Instructions understood by a Microcontroller is known as the Machine Language.

HOW IS MACHINE CODE RELATED TO ASSEMBLY LANGUAGE?

Machine Code (*The language of the machine*)

- Binary Code (bit sequence) that directs the computer to carry out (execute) a pre-defined operation.

  0010 0111 0000 0000

  1110 1111 0001 1111

  1011 1001 0000 0111

  1011 1001 0001 1000



Assembly Language

- A computer language where there is a one-to-one correspondence between a symbolic (assembly language instruction) and a **machine code** instruction.
- *The language of the machine in human readable form*

  clr r16

  ser r17

  out DDRC, r16

  out PORTC, r17

Corollary

- Specific to a single computer or class of computers (non-portable)

2.4. Understanding logic instructions for registers.

Understanding logic instructions for registers, stack operations, and interfacing input and output devices is essential in the field of computer architecture and digital systems design. This knowledge is crucial for

understanding how a computer system processes data and interacts with external devices. In this comprehensive response, we will delve into each of these topics, providing a detailed explanation of their concepts, functions, and operations.

A. Registers

Registers are small, high-speed storage units within a computer's central processing unit (CPU). They store data that the CPU needs to access quickly during its operations. Registers are built using flip-flops or other types of memory elements and are typically organized into different types based on their purpose. There are several types of registers commonly found in computer architectures:

B. General purpose register

These registers are used to store operands, intermediate results, and addresses during program execution.

They can be accessed by the programmer and are often used for arithmetic and logical operations.

C. Special purpose register These registers have specific functions within the CPU. Examples include the program counter (PC), which holds the address of the next instruction to be executed; the instruction register (IR), which holds the current instruction being executed; and the stack pointer (SP), which points to the top of the stack.

D. Control Register: These registers control various aspects of the CPU's behavior, such as interrupt handling, memory management, and cache configuration.

Registers play a vital role in executing instructions efficiently. They act as temporary storage locations for data being processed by the CPU, reducing the need to access slower main memory frequently. Additionally, they enable faster data transfers between different components of the CPU.

E. A stack operation: A stack is a data structure that follows the Last-In-First-Out (LIFO) principle. It is commonly used in computer systems for managing subroutine calls, storing local variables, and handling interrupts. Stack operations involve pushing (adding) or popping (removing) data onto or from the stack.

The stack typically resides in the computer's main memory and grows or shrinks dynamically as data is pushed or popped. It is often implemented using a region of memory called the stack segment, with a stack pointer indicating the current top of the stack.

The two fundamental operations performed on a stack are:

I. Push : This operation adds data to the top of the stack. It involves decrementing the stack pointer and storing the data at the new top position.

II. **Pop:** This operation removes data from the top of the stack. It involves retrieving the data from the current top position and incrementing the stack pointer.

Stack operations are crucial for managing program flow and preserving data integrity during subroutine calls. When a subroutine is called, its return address and local variables are typically pushed onto the stack. Upon completion, these values are popped back to restore the previous execution state.

2.4.1. Instructions of Assembly Language

An instruction in assembly language is a symbolic representation of a single machine instruction. In its simplest form, an instruction consists of a mnemonic and a list of operands. A mnemonic is a short alphabetic code that assists the CPU in remembering an instruction. This mnemonic can be followed by a list of operands. Each instruction in assembly language is coded into one or more bytes. The first byte is generally an OpCode, i.e. a numeric code representing a particular instruction. Additional bytes may affect the action of the instruction or provide information about the data needed by the instruction.

Instruction Semantics:

The following rules have to be strictly followed in order to write correct code.

A. Both operands have to be of the same size:

- **Instruction Correct Reason**

✓ MOV AX, BL No Operands of different sizes

✓ MOV AL, BL Yes Operands of same sizes

✓ MOV AH, BL Yes Operands of same sizes

✓ MOV BL, CX No Operands of different sizes

B. Both operands cannot be memory operands simultaneously:

- **Instruction Correct Reason**

✓ MOV i , j No Both operands are memory variables

✓ MOV AL, i Yes Move memory variable to register

✓ MOV j, CL Yes Move register to memory variable

C. First operand, or destination, cannot be an immediate value:

- **Instruction Correct Reason**

✓ ADD 2, AX No Move register to constant

✓ ADD AX, 2 yes Move constant to register

2.4.2. Logical Instructions

The processor instruction set provides the instructions AND, OR, XOR, TEST and NOT Boolean logic, which tests, sets and clears the bits according to the need of the program.

The format for these instructions

SN	Instruction	Format
1	AND	AND operand1, operand2
2	OR	OR operand1, operand2
3	XOR	XOR operand1, operand2
4	TEST	TEST operand1, operand2
5	NOT	NOT operand1

The first operand in all the cases could be either in register or in memory. The second operand could be either in register/memory or an immediate (constant) value. However, memory to memory operations are not possible.

These instructions compare or match bits of the operands and set the CF, OF, PF, S and ZF flags.

A. The AND Instruction

The AND instruction is used for supporting logical expressions by performing bitwise AND operation. The bitwise

AND operation returns 1, if the matching bits from both the operands are 1, otherwise it returns 0. For example

```
Operand1:    0101
Operand2:    0011
----------------------------
After AND -> Operand1:    0001
```

The AND operation can be used for clearing one or more bits. For example, say, the BL register contains 0011

1010. If you need to clear the high order bits to zero, you AND it with 0FH.

```
AND     BL,   0FH    ; This sets BL to 0000 1010
```

Let's take up another example. If you want to check whether a given number is odd or even, a simple test would be to check the least significant bit of the number. If this is 1, the number is odd,

else the number is even.

Assuming the number is in AL register, we can write:

```
AND     AL, 01H     ; ANDing with 0000 0001
JZ      EVEN_NUMBER
```

The following program illustrates this:

Example:

```
section     .text
global  main                    ;must be declared for using gcc
main:                           ;tell linker entry point
mov     ax,     8h              ;getting 8 in the ax
and     ax, 1                   ;and ax with 1
jz      evnn
mov     eax, 4                  ;system call number (sys_write)
mov     ebx, 1                  ;file descriptor (stdout)
mov     ecx, odd_msg            ;message to write
mov     edx, len2               ;length of message
int     0x80                    ;call kernel
jmp     outprog
evnn:
mov     ah, 09h
mov     eax, 4                  ;system call number (sys_write)
mov     ebx, 1                  ;file descriptor (stdout)
mov     ecx, even_msg           ;message to write
mov     edx, len1               ;length of message
int     0x80                    ;call kernel
outprog:
mov     eax,1                   ;system call number (sys_exit)
int     0x80                    ;call kernel
section     .data
even_msg  db  'Even Number!' ;message showing even number
len1  equ  $ - even_msg
```

```
odd_msg db 'Odd Number!' ;message showing odd number
len2 equ $ - odd_msg
```

When the above code is compiled and executed, it produces following result:

```
Even Number!
```

Change the value in the ax register with an odd digit, like:

```
mov ax, 9h ; getting 9 in the ax
```

The program would display:

```
Odd Number!
```

Similarly to clear the entire register you can AND it with 00

B. The OR Instruction

The OR instruction is used for supporting logical expression by performing bitwise OR operation. The bitwise OR Operator returns 1, if the matching bits from either or both operands are one. It returns 0, if both the bits are zero.

For example,

```
Operand1: 0101
Operand2: 0011
----------------------------
```

The OR operation can be used for setting one or more bits. For example, let us assume the AL register contains

0011 1010, you need to set the four low order bits, you can OR it with a value 0000 1111, i.e., FH.

```
OR BL, 0FH ; This sets BL to 0011 1111
```

Example:

The following example demonstrates the OR instruction. Let us store the value 5 and 3 in the AL and the BL register
respectively. Then the instruction,

```
OR AL, BL
```

should store 7 in the AL register:

```
section .text
global main ;must be declared for using gcc
main: ;tell linker entry point
mov al, 5 ;getting 5 in the al
mov bl, 3 ;getting 3 in the bl
or al, bl ;or al and bl registers, result should be 7
add al, byte '0' ;converting decimal to ascii
mov [result], al
mov eax, 4
mov ebx, 1
mov ecx, result
mov edx, 1
int 0x80

outprog:
mov eax,1 ;system call number (sys_exit)
int 0x80 ;call kernel
section .bss
result resb 1
```

When the above code is compiled and executed, it produces following result:

C. The XOR Instruction

The XOR instruction implements the bitwise XOR operation. The XOR operation sets the resultant bit to 1, if and
only if the bits from the operands are different. If the bits from the operands are same (both 0

or both 1), the
resultant bit is cleared to 0.
For example,

```
Operand1:       0101
Operand2:       0011
---------------------------
After  XOR  ->  Operand1:      0110
```

XORing an operand with itself changes the operand to 0. This is used to clear a register.

```
XOR      EAX,  EAX
```

D. The TEST Instruction

The TEST instruction works same as the AND operation, but unlike AND instruction, it does not change the first operand. So, if we need to check whether a number in a register is even or odd, we can also do this using the TEST instruction without changing the original number.

```
TEST     AL,  01H
JZ       EVEN_NUMBER
```

E. The NOT Instruction

The NOT instruction implements the bitwise NOT operation. NOT operation reverses the bits in an operand. The operand could be either in a register or in the memory.
For example,

```
Operand1:       0101  0011
After  NOT  ->  Operand1:      1010  1100
```

2.4.3. Assembly Conditions

Conditional execution in assembly language is accomplished by several looping and branching instructions. These instructions can change the flow of control in a program. Conditional execution is observed in two scenarios:

A. Conditional Instructions

- **B. Unconditional jump**

This is performed by the JMP instruction. Conditional execution often involves a transfer of control to the address of an instruction that does not follow the currently executing instruction. Transfer of control may be forward to execute a new set of instructions, or backward to re-execute the same steps.

- **Conditional jump**

This is performed by a set of jump instructions j<condition> depending upon the condition. The conditional instructions transfer the control by breaking the sequential flow and they do it by changing the offset value in

Let us discuss the CMP instruction before discussing the conditional instructions.

B. The CMP Instruction

The CMP instruction compares two operands. It is generally used in conditional execution. This instruction basically subtracts one operand from the other for comparing whether the operands are equal or not. It does not disturb the destination or source operands. It is used along with the conditional jump instruction for decision

making.

SYNTAX

- **CMP destination, source**

CMP compares two numeric data fields. The destination operand could be either in register or in memory. The source operand could be a constant (immediate) data, register or memory.

EXAMPLE:

```
CMP DX, 00  ; Compare the DX value with zero
JE  L7      ; If yes, then jump to label L7
```

CMP is often used for comparing whether a counter value has reached the number of time a loop needs to be run. Consider the following typical condition:

```
INC   EDX
CMP   EDX, 10 ; Compares whether the counter has reached 10
JLE   LP1     ; If it is less than or equal to 10, then jump to LP1
```

- **Unconditional Jump**

As mentioned earlier this is performed by the JMP instruction. Conditional execution often involves a transfer of control to the address of an instruction that does not follow the currently executing instruction. Transfer of control may be forward to execute a new set of instructions, or

backward to re-execute the same steps.

- **SYNTAX:**

The JMP instruction provides a label name where the flow of control is transferred immediately. The syntax of the

JMP instruction is:

```
JMP     label
```

EXAMPLE:

The following code snippet illustrates the JMP instruction:

```
MOV  AX, 00    ; Initializing AX to 0
MOV  BX, 00    ; Initializing BX to 0
MOV  CX, 01    ; Initializing CX to 1
L20:
ADD  AX, 01    ; Increment AX
ADD  BX, AX    ; Add AX to BX
SHL  CX, 1     ; shift left CX, this in turn doubles the CX value
JMP  L20       ; repeats the statements
```

- **Conditional Jump**

If some specified condition is satisfied in conditional jump, the control flow is transferred to a target instruction.

There are numerous conditional jump instructions, depending upon the condition and data.

Following are the conditional jump instructions used on signed data used for arithmetic operations:

Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JG/JNLE	Jump Greater or Jump Not Less/Equal	OF, SF, ZF
JGE/JNL	Jump Greater or Jump Not Less	OF, SF
JL/JNGE	Jump Less or Jump Not Greater/Equal	OF, SF
JLE/JNG	Jump Less/Equal or Jump Not Greater	OF, SF, ZF

Following are the conditional jump instructions used on unsigned data used for logical operations:

Instruction	Description	Flags tested

2.4.4. Write an assembly language program

These are the steps that should be followed for writing an assembly language program:

- ✓ Define the problem.
- ✓ Write the algorithm.
- ✓ Translate into assembly mnemonics.
- ✓ Test and Debug the program in case of errors.

- **The translation phase consists of the following steps:**
  - ✓ Define type of data the program will deal with.
  - ✓ Write appropriate instructions to implement the algorithm.

2.4.5. Development Tools For Assembly Language Program

Now that you have some idea, about how to go about writing assembly language programs, you might want to write your own programs, and try them out on the machine. To do that, there are some developmental tools required. Let us study them now. The discussion is from the point of view of the end user, and not the system programmer.

A. Editor

An editor is a program which, when run on a system, lets you type in text, and store in a file. This text could also be your assembly language program. There are a number of editors available on PC. The editor helps you type the program in required format. This form of the program is called as the source program. The editor gives you all the flexibility, to insert lines, delete lines, insert words, characters, delete words, characters etc. In short all the features that you can think of while writing text, and more. After the program is typed, it can be stored in some secondary storage, like hard disk, floppy diskette etc, for permanent storage.

B. Assembler

An assembler program is used to translate assembly language mnemonics to the binary code for each instruction. After the complete program has been written, with the help of an editor, it is then assembled with the help of an assembler. An assembler works in two phases, i.e., it reads your source code two times. In the first pass, the assembler, collects all the symbols defined in the program, along

with their offsets, in symbol table. On a second pass through the source program, it produces a binary code for each instruction of the program, and give all the symbols an offset with respect to the segment, from the symbol table.

The assembler generates two files: the object file and the list file. The object file contains the binary code for each instruction in the program. It is created only when your program has been successfully assembled, with no errors. The errors that are detected by the assembler, are called the syntax errors. These are like:

- MOVE AX,BX ; undeclared identifier MOVE.
- MOV AX,BL ; illegal operands

These are just two of the syntax errors that you can get when your program contains such kind of mistakes. (Exact description of the errors defer from assembler to assembler). In the first statement, it reads the word MOVE, it tries to match with its mnemonics set, as there is no mnemonic with this spelling, it assumes it to be an identifier, and looks for its entry in the symbol table. It does not even find it there, therefore, gives an error 'undeclared identifier'. In the second error, the two operands are of different kind. 8086 expects, both the identifier to be of the same kind, byte or

word. But in the above case, one is a byte variable, while the other is a word variable. An assembler does not detect logical errors in your programs, that is your responsibility. List file is optional, and contains, the source code, the binary equivalent of each instruction, and the offsets of the symbols in the program. This file is for documentation purposes. Some of the assemblers available on PC are, MASM (Microsoft Assembler), TASM (TURBO) etc.

C. Linker

For modularity of your program, it is better to break your programs, into several subroutines. It is even better, to put the common routine, like reading a hexadecimal number, writing a hexadecimal number etc., which could he used by a lot of your other programs also, into a separate file. These files are assembled separately. After each ,has been successfully assembled, they can be linked together to form a large file, which constitutes your complete program. The file containing the common routines can be linked to your other programs also. The program that links your programs is called the linker.

The linker produces a link file which contains the binary codes for all compound modules. The linker also produces a link map which contains the address information about the linked files. The linker, however, does not assign absolute addresses to your program. It only assigns continuous relative addresses to all the modules linked, starting from zero. This form of program is said to be reloadable, because it

can be put anywhere in memory to be run. This form of code can be even be carried to other machines, of the same kind, or compatible to the present machine, to be run successfully. The linker available on your PC is LINK ,TURBO has a built in linker.

D. Loader

Loader is a program, which assigns absolute addresses to the program. These addresses are generated, by adding to all the offsets, the address from where the program is loaded into the memory. Loader comes into action, when you execute your program. This program is brought from the secondary memory, like disk, or floppy diskette, into the main memory at a specific address. Let us assume the program was loaded at address 1000h, then 1000h is added to all the offsets to get the absolute address. Once the program has been loaded, it is now ready to run.

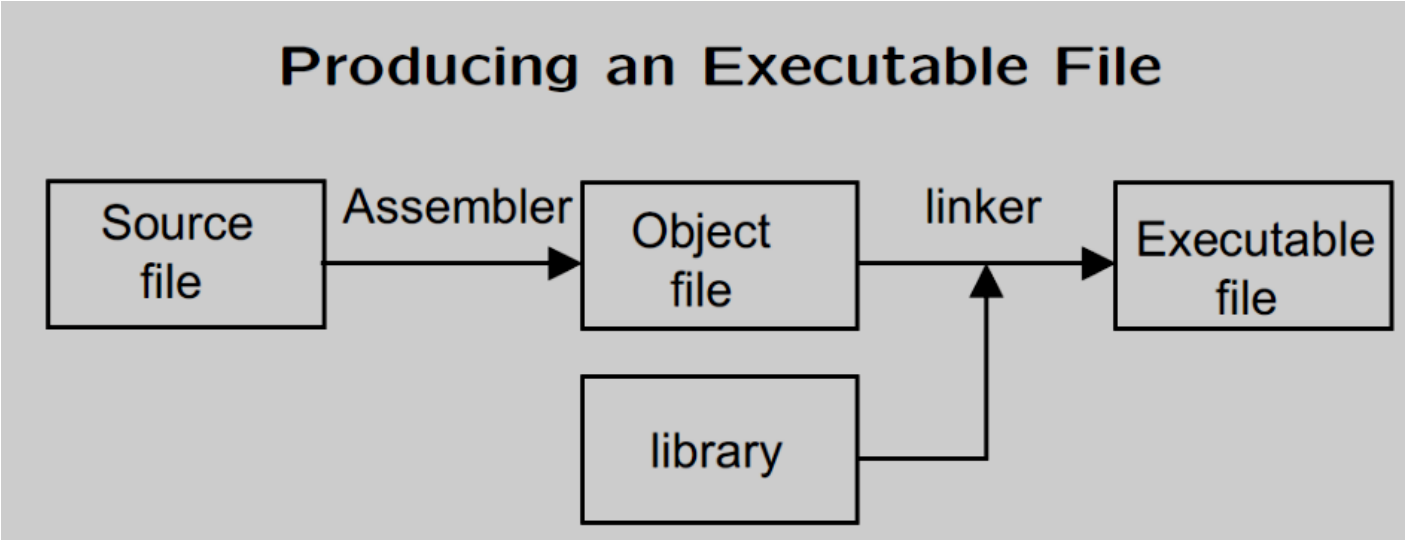
E. Debugger

If your program requires no external hardware or requires hardware directly accessible from your system, then you can use a debugger to debug your program. Debugger allows you to load your program into just like a loader, and, troubleshoot your program. While debugging, you can run your program in single step, set breakpoints, and view the contents of registers or memory locations. You can even change the contents of the register or memory location, and run your program with new value. This helps you to isolate the problems in your programs. The problems can be

Corrected with the help of an editor, and the whole procedure of assembling, linking and executing your program can be repeated. Debugger helps you detect the logical errors that could not be detected by the assembler.

- **The following steps showed the process of translating an assembly program into Executable file:**

✓ The assembler produces an object file from the assembly language source.
✓ The object file contains machine language code with some external and re-locatable
✓ addresses that will be resolved by the linker. Their values are undetermined at that stage.
✓ The linker extract object modules (compiled procedures) from a library and links
✓ them with the object file to produce the executable file.
✓ The addresses in the executable file are all resolved but they are still virtual addresses.



2.4.6. Interfacing Input and output (I/O) devices

Input and output (I/O) devices allow computers to interact with the external world by receiving input from users and providing output to display information or control external devices. Interfacing these devices with a computer system involves establishing communication channels and protocols to exchange data. There are various methods for interfacing I/O devices, including:

A. Memory mapped device :

In this method, I/O devices are assigned specific memory addresses. Reading from or writing to these addresses triggers corresponding actions on the devices. The CPU communicates with I/O devices by reading from or writing to specific memory locations.

B.Port mapped I/O : In this method, separate input/output ports are dedicated to each device. The CPU communicates with I/O devices by reading from or writing to specific port addresses.

C. Direct memory access (DMA) : DMA allows certain devices to transfer data directly between their own memory and main memory without CPU intervention. This method is particularly useful for high-speed data transfers, such as disk I/O or network communication.

Interfacing input devices involves reading data from sources such as keyboards, mice, sensors, or network interfaces. Output devices, on the other hand, receive data from the computer system and display it on screens, printers, speakers, or other output mediums.

Effective interfacing requires understanding the protocols and communication standards specific to each

device. These standards can include serial or parallel communication, USB, Ethernet, HDMI, or wireless protocols like Bluetooth or Wi-Fi.

In conclusion, understanding logic instructions for registers, stack operations, and interfacing input and output devices is crucial for comprehending computer architecture and digital systems design. Registers provide fast storage within the CPU for efficient data processing. Stack operations enable proper management of subroutine calls and local variables. Interfacing input and output devices allows computers to interact with the external world effectively.

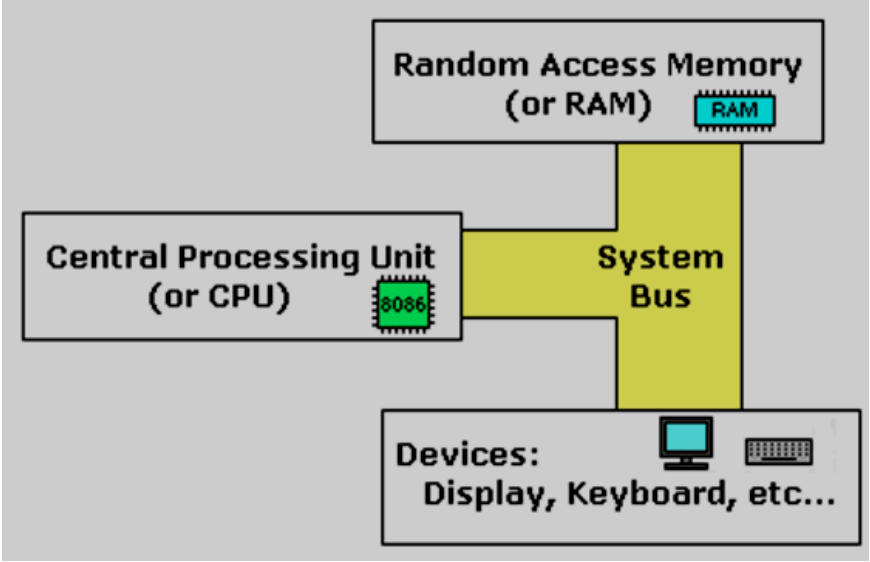
2.4.7. Assembly language

Assembly language is a low-level programming language that is specific to a particular microprocessor or microcontroller. It is used to write programs that can directly communicate with the hardware of microprocessor-based equipment. In this response, we will discuss the process of preparing assembly language code for microprocessor-based equipment, including the steps involved and the resources available in PDF and DOC formats.

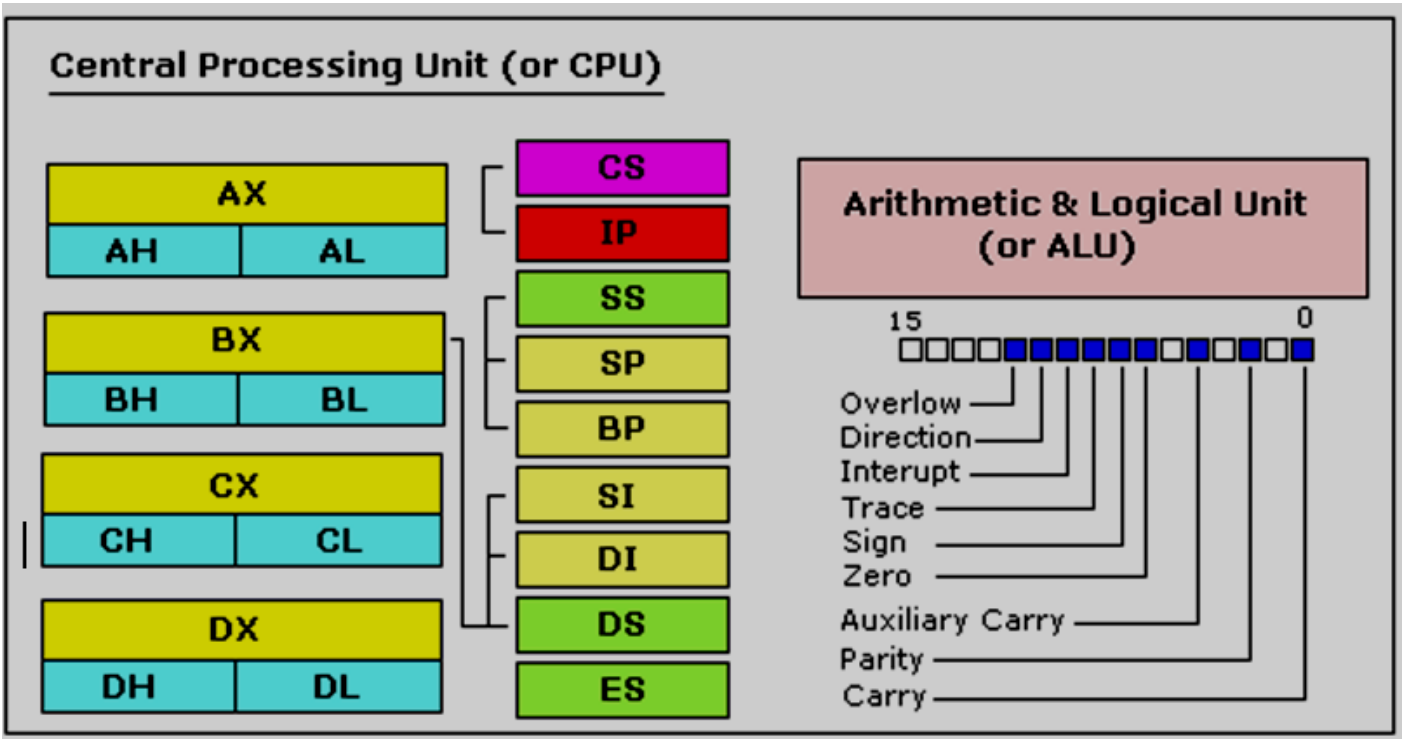
What is Assembly language?

Assembly language is a low level programming language. you need to get some knowledge about computer structure in order to understand anything. The simple computer model as I seeit: the **system bus** (shown in yellow) connects the various components of a computer. the **CPU** is the heart of the computer, most of computations occur inside the **CPU**.

RAM is a place to where the programs are loaded in order to be executed.



- **Inside the CPU**



- **General Purpose Registers**

8086 CPU has 8 general purpose registers; each register has its own name:

- **AX** - the accumulator register (divided into **AH / AL**).
- **BX** - the base address register (divided into **BH / BL**).
- **CX** - the count register (divided into **CH / CL**).
- **DX** - the data register (divided into **DH / DL**).
- **SI** - source index register.
- **DI** - destination index register.
- **BP** - base pointer.
- **SP** - stack pointer.

despite the name of a register, it's the programmer who determines the usage for each general purpose register. the main purpose of a register is to keep a number (variable). the size of the above registers is 16 bit, it's something like: **0011000000111001b** (in binary form), or **12345** in decimal (human) form. **4 general purpose registers** (AX, BX, CX, DX) are made of two separate 8 bit registers, for example if AX= **0011000000111001b**, then AH=**00110000b** and AL=**00111001b**. therefore, when you modify any of the 8 bit registers 16 bit register is also updated, and vice-versa. the same is for other 3 registers, "H" is for high and "L" is for low part.

because registers are located inside the CPU, they are much faster than memory. Accessing a memory location requires the use of a system bus, so it very small and most registers have special purposes which limit their use as variables, but they are still an excellent place to store temporary data of calculations.

Segment Registers

- **CS** - points at the segment containing the current program.

DS - generally points at segment where variables are defined.

- **ES** - extra segment register, it's up to a coder to define its usage.
- **SS** - points at the segment containing the stack. although it is possible to store any data in the

segment registers, this is never a good idea. the segment registers have a very special purpose - pointing at accessible blocks of memory. segment registers work together with general purpose register to access any memory value. For example if we would like to access memory at the physical address **12345h** (hexadecimal), we should set the **DS = 1230h** and **SI = 0045h**. This is good, since this way we can access much more memory than with a single register that is limited to 16 bit values. CPU makes a calculation of physical address by multiplying the segment register by 10h and adding general purpose register to it (1230h * 10h + 45h

= 12345h):

```
 12300
+ 0045
-------
 12345
```

The address formed with 2 registers is called an **effective address**. by default **BX, SI** and **DI** registers work with **DS** segment register; **BP** and **SP** work with **SS** segment register. other general purpose registers cannot form an effective address! also, although **BX** can form an effective address, **BH** and **BL** cannot.

- **Special Purpose Registers**
    - ✓ **IP** - the instruction pointer.
    - ✓ **Flags register** - determines the current state of the microprocessor.

IP register always works together with **CS** segment register and it points to currently executing instruction.

Flags register is modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. Generally you cannot access these registers directly, the way you can access AX and other general registers, but it is possible to change values of system registers using some tricks that you will learn a little bit later.

2.4.8. Preparing assembly language code for microprocessor-based equipment involves

A. several steps

- **Understanding the Microprocessor:** The first step in preparing assembly language code is to have a thorough understanding of the microprocessor or microcontroller being used. This includes knowing its architecture, instruction set, memory organization, and I/O capabilities. Different microprocessors have different instruction sets and addressing modes, so it is important to be familiar with these details before writing assembly code.
- **Defining the Requirements:** Once you understand the microprocessor, you need to define the requirements of the program you want to write. This includes identifying the inputs and outputs of the program, as well as any specific functionality or algorithms that need to be implemented. By clearly defining the requirements, you can plan your assembly code accordingly.
- **Writing the Assembly Code:** After defining the requirements, you can start writing the assembly code. Assembly language uses mnemonic instructions that correspond to machine instructions understood by the microprocessor. These instructions are written in a human-readable format and are then translated into machine code using an assembler. The assembly code should be structured and well-commented for better readability and maintainability.
- **Testing and Debugging:** Once you have written the assembly code, it is important to thoroughly test and debug it. This involves running the program on the target microprocessor-based equipment and verifying that it produces the expected results. Testing should cover various scenarios and edge cases to ensure that the program functions correctly under different conditions.

- **Optimization:** After testing and debugging, you can optimize the assembly code to improve its performance or reduce its size. This can involve techniques such as loop unrolling, code rearrangement, or using specific instructions provided by the microprocessor for efficient execution. Optimization should be done carefully, considering the trade-offs between code size, execution speed, and maintainability.
- **Documentation:** Finally, it is important to document the assembly language code for future reference and maintenance. This includes providing a clear description of the program's functionality, any assumptions made during development, and any specific hardware dependencies or configurations required. Documentation helps other developers understand and modify the code in the future.

B. Preparing assembly language code for microprocessor-based equipment

- **"The Art of Assembly Language Programming" by Randall Hyde:** This book provides a comprehensive introduction to assembly language programming and covers various aspects such as data representation, memory management, and I/O operations. It also includes examples and exercises to reinforce the concepts discussed.
- **"Microprocessor Architecture, Programming, and Applications with the 8085" by Ramesh Gaonkar:** specifically focuses on the Intel 8085 microprocessor architecture and programming. It covers topics such as instruction set architecture, addressing modes, interrupts, and I/O interfacing. The book includes numerous examples and programming exercises to enhance understanding.
- **Intel 80x86 Assembly Language Programming Manual:** Intel provides detailed documentation for their 80x 86 families of microprocessors, including manuals that cover assembly language programming. These manuals provide information about instruction set architecture, addressing modes, memory organization, and other relevant details specific to Intel microprocessors. These authoritative reference publications provide valuable insights into assembly language programming for microprocessor-based equipment. They offer comprehensive explanations of concepts and practical examples that can assist programmers in writing efficient and effective assembly code.

1.5. Preparing assembly language

To prepare assembly language instructions for microprocessor-based equipment verification and entering them into a PDF or DOC file, you will need to follow a series of steps. Assembly language is a low-level programming language that is specific to the architecture of a particular microprocessor. It allows direct control over the hardware and provides a more efficient way to write programs compared to higher-level languages.
Here is a comprehensive guide on how to prepare assembly language instructions for microprocessor-based equipment verification and enter them into file:

- **Understand the Microprocessor Architecture:** The first step is to familiarize yourself with the architecture of the microprocessor you are working with. Each microprocessor has its own instruction set, registers, memory organization, and addressing modes. You need to understand these details to write efficient and correct assembly code.
- **Choose an Assembly Language Development Environment:** There are several assembly language development environments available, such as Microsoft Macro Assembler (MASM), GNU Assembler (GAS),

and NASM (Netwide Assembler). Select an environment that supports the microprocessor you are using and install it on your computer.

- **Write Assembly Language Instructions:** Start by writing the assembly language instructions using a text editor or an integrated development environment (IDE) provided by the chosen development environment. Assembly language instructions consist of mnemonics (operation codes) and operands (registers, memory addresses, constants) that specify the desired operation.
- **Verify the Instructions:** After writing the assembly language instructions, it is crucial to verify their correctness. This can be done through manual inspection or by using a debugger provided by the development environment. Step through the code and ensure that each instruction performs the intended operation.
- **Assemble the Code:** Once you have verified the instructions, assemble them into machine code using the assembler provided by your chosen development environment. The assembler converts assembly language instructions into binary machine code that can be executed by the microprocessor.
- **Create a PDF or DOC File:** To enter the assembly language instructions into a PDF or DOC file, you can use a word processing software like Microsoft Word or Google Docs. Open a new document and copy-paste the assembly code into it. Format the code for readability by using appropriate indentation and syntax highlighting.
- **Save the Document:** Save the document as a PDF or DOC file format. In Microsoft Word, you can choose the "Save As" option and select either PDF or DOC as the desired file format. In Google Docs, go to "File" > "Download" and choose either PDF or DOC.
- **Organize and Document:** It is essential to organize your assembly language instructions in a logical manner and provide appropriate documentation. Use comments within the code to explain the purpose of each instruction, any assumptions made, and any limitations or dependencies.

- **Test and Debug:** Before finalizing your assembly language instructions, thoroughly test them on the target microprocessor-based equipment. Use appropriate test cases to ensure that the program behaves as expected. If any issues arise, debug the code using the debugger provided by your chosen development environment.
- **Finalize and Distribute:** Once you are confident in the correctness of your assembly language instructions, finalize the document by proofreading it for any errors or typos. Ensure that all necessary information is included, such as contact details, version numbers, and revision history. Finally, distribute the PDF or DOC file to relevant stakeholders or users.

- In conclusion, preparing assembly language instructions for microprocessor-based equipment verification and entering them into a PDF or DOC file involves understanding the microprocessor architecture, choosing an assembly language development environment, writing the instructions, verifying their correctness, assembling the code, creating a PDF or DOC file using word processing software, organizing and documenting the code, testing and debugging it on the target equipment, finalizing the document, and distributing it to stakeholders.

1.6. Implementing basic software technique

A. **execution of Software package**

"**Software package"** is a commonly used term; however, its general, widespread usage has resulted in different meanings. The origins of software packages can be traced to the 1960s. The Program Application Library on the IBM 1400 series computer, the MARK IV package by Informatics, and the Auto flow package by Applied Data Research were some of the first software packages developed. What made them unique was that one of the goals of each of these packages was to be usable (or transportable) on a number of different types of computers [WELK80].

One of the most significant events in the evolution of software packages was IBM's decision in 1969 to "unbundle", or offer software separately from hardware, training and documentation. Software products became recognized as distinct entities, separately priced from computer hardware. Most of the initially unbundled software products or packages were system software packages. These packages were intended primarily to enhance the performance of the ADP system and included, for example, compilers, sort utilities, and data base management systems.

Prompted at least in part by the increasingly widespread use of microcomputers, the recent explosion in both the number and popularity of software packages has resulted in a number of distinct classes of packages. Software tools aid in the development of computer programs; systems packages analyze or improve the performance of a component of the system; data base management systems organize, interrogate, and update data files. Application software packages are the class of software packages considered in this document. They both directly and indirectly support functional managers, concerned with such areas as personnel and project management, administration, inventory control, budget and financial management, word processing and office automation, and support functions typically performed in most organizations, whether Government or private sector. Specialized functions such as process control, airline reservation systems and military embedded weapon systems are not included in this definition. Systems - 9 - software packages and software tools are also not included. Another key in this definition of

software packages is that the software is developed with the intent to sell the package to a number of customers. This implies that the package has already been developed and is offered for immediate delivery. This type of package is often called off-the-shelf software, implying that no further development is required for the package to be used.

The majority of application software packages share the following characteristics;

o They are frequently leased or licensed, rather than sold.

o They are usually provided in object code form and restricted to certain computer systems for use. o User documentation is provided with the package. o Maintenance of the package is separately priced, but a warranty period is often provided. o Training is provided, perhaps separately priced. o Updates to the package are expected. o Installation of the package is sometimes provided by the vendor . 2.2 Applications Software packages support an increasing number of functions within the business environment. Some of the major classes of packages used by private industry, and available for Federal government consideration, are discussed in this section. Not all applications are included. Rather, the following discussions are intended to illustrate the types and range of functions provided by available packages.

1.6.1. Detail command based on the logic instructions.

In microprocessor-based equipment, command execution is carried out through a series of logic instructions. These instructions are stored in memory and are fetched and executed by the microprocessor to perform specific tasks. The commands are typically written in assembly language or machine code, which is a low-level programming language understood by the microprocessor.

There are various types of logic instructions that can be used in microprocessor-based equipment, depending on the specific architecture and instruction set of the microprocessor.

A. Some common logic instructions include:

- **Arithmetic Instructions:** These instructions perform basic arithmetic operations such as addition, subtraction, multiplication, and division. They operate on data stored in registers or memory locations and store the result back in a register or memory.
- **Logical Instructions:** Logical instructions perform logical operations such as AND, OR, XOR, and NOT on binary data. These instructions are used for tasks like bit manipulation, data masking, and Boolean operations.
- **Data Transfer Instructions:** Data transfer instructions move data between registers, memory locations, and input/output devices. They can be used to load data into registers, store data in memory, or transfer data between different memory locations.

- **Control Transfer Instructions**: Control transfer instructions alter the flow of program execution based on certain conditions. They include conditional branch instructions that allow the program to jump to a different location in memory depending on the result of a previous operation or a specific condition.
- **Input/output Instructions:** These instructions enable communication between the microprocessor and external devices such as sensors, actuators, displays, or communication interfaces. They allow data to be transferred between the microprocessor and these devices.
- **Stack Instructions:** Stack instructions are used to manage a stack data structure in memory. They include push and pop operations that add or remove data from the top of the stack.
- **Interrupt Instructions:** Interrupt instructions handle interrupts generated by external events or internal conditions. When an interrupt occurs, the microprocessor suspends its current execution and jumps to a specific interrupt handler routine to handle the event.
- **System Control Instructions:** These instructions control various system-level operations such as power management, clock configuration, and memory protection.

The specific command syntax and format for logic instructions depend on the microprocessor architecture and instruction set. Each microprocessor has its own set of instructions with unique mnemonics and operand formats. Therefore, it is essential to refer to the documentation or technical manuals provided by the microprocessor manufacturer to obtain the precise command details for particular microprocessor-based equipment

1.6.2. Set up of STO, HALT, and FLAG to set or reset for given data

- To set or reset STO, HALT, and FLAG for a given data PDF or DOC file, you would need to understand the specific context in which these terms are being used. Without further clarification, it is difficult to provide a precise answer. However, I can explain the general concepts of STO, HALT, and FLAG.
- **STO (Set/Store):** STO typically refers to the "store" operation in computer programming or digital systems. It is used to save or store data into a specific memory location or register. In the context of a PDF or DOC file, STO could be used to indicate saving or storing certain information within the file itself. However, it is important to note that PDF and DOC files are binary formats with their own internal structures, so directly manipulating individual bits or bytes may not be straightforward.

- **HALT:** HALT generally refers to the "halt" instruction in computer architecture or programming languages. It is used to stop the execution of a program or process. In the context of a PDF or DOC file, HALT could be used metaphorically to indicate the termination of a specific operation or process related to the file. For example, if there was an ongoing conversion process from PDF to DOC format, issuing a HALT command would terminate that conversion.

- **FLAG:** FLAG is a term commonly used in computer programming to represent a binary value that indicates a certain condition or state. Flags are often used for control flow decisions and conditional branching within programs. In the context of a PDF or DOC file, FLAG could be used to represent various states or conditions related to the file's content or metadata. For example, a FLAG could be set to indicate whether a particular section of the document has been reviewed or not.

ADDRESSING MODES

MC6802 Microprocessor has 7 addressing modes that can be used by the programmer:

1. Accumulator
2. Immediate
3. Direct
4. Extended
5. Indexed
6. Implied (Inherent)
7. Relative

MC6802 instructions may be used with one or more of these addressing modes. The instruction set and their addressing modes are given in Appendix A.

Accumulator Addressing

In accumulator addressing, either accumulator A or accumulator B is specified. These are 1- byte instructions. **Ex: ABA** adds the contents of accumulators and stores the result in accumulator A

Immediate Addressing

In immediate addressing, operand is located immediately after the opcode in the second byte of the instruction in program memory (except LDS and LDX where the operand is in the second and third bytes of the instruction). These are 2-byte or 3-byte instructions.

Direct Addressing

In direct addressing, the address of the operand is contained in the second byte of the instruction. Direct addressing allows the user to directly address the lowest 256 bytes of the memory, i.e, locations 0 through 255. Enhanced execution times are achieved by storing data in these locations.

These are 2-byte instructions. **Ex: LDAA 25H** loads the contents of the memory address $(25)_H$ into accumulator A

Extended Addressing

In extended addressing, the address contained in the second byte of the instruction is used as the higher eight bits of the address of the operand. The third byte of the instruction is used as the lower

eight bits of the address for the operand. This is an absolute address in the memory. These are 3-byte

instructions. **Ex: LDAA 1000H** loads the contents of the memory address $(1000)_H$ into accumulator A

Self Check 2.1.

TEST I: <u>Say True or False</u>

1. HALT is used to stop the execution of a program or process
2. **Command execution** is carried out through a series of logic instructions.
3. Control transfer instructions alter the flow of program execution based on certain conditions.
4. Stack instructions are used to manage a stack data structure in memory
5. Data transfer instructions move data between registers, memory locations, and input/output devices.

TEST II: <u>Chose the best answer</u>

1. **Which one of the following is not special Purpose Registers**

A. IP B. **Flags register** **C. 8086 CPU** **D. None**

TEST III : <u>Write Short answer</u>

1. Define segment Registers

2. List the 8 elements of 8086 CPU general Purpose Registers

3. Prepare assembly language instructions for microprocessor-based equipment verification a

Unit Three: Assemble and enter operating instructions

This learning unit is developed to provide the trainees the necessary information regarding the following content coverage and topics:

- Checking and isolating circuits/machines/plant.
- Identifying microprocessors equipped devices
- Checking entered data.
- Responding unplanned situations

This unit will also assist you to attain the learning outcomes stated in the cover page. Specifically, upon completion of this learning guide, you will be able to:

- Ensure and isolate circuits/machines/plant.
- Identify microprocessors equipped devices
- Confirm entered data.
- Respond unplanned situations

3.1. Checking and isolating circuits/machines/plant.

When it comes to checking and isolating circuits, machines, or plants for microprocessor equipment, there are several important steps and considerations to keep in mind. This process is crucial for troubleshooting, maintenance, and ensuring the smooth operation of microprocessor-based systems. In this comprehensive response, we will delve into the various aspects of checking and isolating circuits and provide a detailed guide on how to approach this task effectively.

3.1.1. Understanding Microprocessor Equipment

Microprocessor equipment refers to devices or systems that utilize microprocessors as their central processing units (CPUs). These microprocessors are integrated circuits that execute instructions and perform calculations in electronic devices. They are commonly found in computers, smartphones, industrial control systems, and many other electronic devices.

Microprocessor-based systems consist of various components such as memory modules, input/output (I/O) interfaces, power supplies, and peripheral devices. These components work together to enable the functioning of the

microprocessor and the overall system. When troubleshooting or maintaining microprocessor equipment, it is essential to understand the interconnections between these components and how they interact with the microprocessor

A. Reasons for checking and isolating circuits/machine/plants

Checking and isolating circuits, machines, or plants with microprocessor equipment is necessary for several reasons:

- **Troubleshooting:** When a microprocessor-based system malfunctions or exhibits unexpected behavior, checking and isolating circuits can help identify the root cause of the issue. By systematically examining different components and their connections, faulty elements can be identified and repaired or replaced.
- **Maintenance:** Regular maintenance is crucial for ensuring the optimal performance and longevity of microprocessor equipment. Checking circuits allows technicians to detect any potential issues before they escalate into major problems. It also helps in identifying areas that require preventive maintenance or component replacement.
- **Safety:** Isolating circuits or machines is essential for safety purposes during maintenance or repair work. By disconnecting power sources and isolating specific circuits, technicians can prevent accidental electrical shocks or damage to the equipment.

B. Steps of checking and isolating circuits/machine/plants

To effectively check and isolate circuits, machines, or plants with microprocessor equipment, the following steps can be followed:

- **Reviewing Documentation:** Start by reviewing the documentation provided by the equipment manufacturer. This includes user manuals, technical specifications, circuit diagrams, and any other relevant documentation. Understanding the system architecture and component interconnections is crucial before proceeding with checking and isolating circuits.
- **Identify Power Sources:** Identify all power sources supplying electricity to the microprocessor equipment. This includes both primary power sources (e.g., main power supply) and secondary power sources (e.g., batteries or backup power supplies). Ensure that all power sources are disconnected or switched off before proceeding further.

- **Disconnect External Connections:** Disconnect any external connections to the microprocessor equipment, such as network cables, peripheral devices, or communication interfaces. This step ensures that external factors do not interfere with the checking and isolation process.
- **Perform Visual Inspection:** Conduct a visual inspection of the equipment to identify any visible signs of damage, loose connections, or burnt components. Look for physical abnormalities that might indicate potential issues.
- **Check Power Supply Units:** Examine the power supply units (PSUs) connected to the microprocessor equipment. Ensure that they are functioning correctly and providing stable power output within the specified voltage range. Use appropriate testing tools like multi-meters to measure voltage levels accurately.
- **Inspect Circuit Boards:** Inspect all circuit boards within the microprocessor equipment for any signs of damage or component failure. Look for burnt components, bulging capacitors, loose connectors, or any other abnormalities. If possible, compare the circuit boards against their respective diagrams to ensure proper component placement.
- **Test Memory Modules:** Memory modules play a crucial role in microprocessor-based systems. Test the memory modules to ensure they are functioning correctly. This can be done using diagnostic software or specialized memory testing tools.
- **Check I/O Interfaces:** Verify the functionality of input/output (I/O) interfaces connected to the microprocessor equipment. Test each interface individually to ensure proper communication and data transfer between the microprocessor and external devices.
- **Isolate Circuits/Machines/Plants:** If a specific circuit, machine, or plant is suspected to be causing issues, isolate it from the rest of the system. This can be done by disconnecting relevant connectors, switches, or relays. By isolating the problematic element, further testing and troubleshooting can be conducted without affecting the overall system.
- **Perform Functional Tests:** After completing the above steps, reconnect power sources and external connections (if necessary) and perform functional tests on the microprocessor equipment. Monitor its behavior and check for any recurring issues or abnormal operation.
- **Document Findings:** Throughout the checking and isolation process, document all findings, including observations, test results, and any actions taken. This documentation serves as a reference for future troubleshooting or maintenance activities.

3.2. Identifying microprocessors equipped devices

Microprocessors are integrated circuits that serve as the central processing unit (CPU) of a computer or other electronic devices. They are responsible for executing instructions and performing calculations, making them a crucial component in various devices. Identifying microprocessor-equipped devices can be done by examining the instruction set architecture (ISA) of the microprocessor, which defines the set of instructions that the microprocessor can execute.

A. To identify microprocessors equipped devices using the instruction set architecture, one can refer to the documentation provided by the microprocessor manufacturer. This documentation is typically available in the form of PDF or DOC files, which contain detailed information about the microprocessor's features, specifications, and instruction set.

B. **To begin the identification process, one needs to obtain the documentation** for the specific microprocessor model used in the device. The documentation can usually be found on the manufacturer's website or through other reliable sources. Once obtained, the PDF or DOC file can be opened and searched for relevant information.

C. **The first step is to locate the section that describes the instruction set architecture**. This section provides an overview of the instructions supported by the microprocessor and their corresponding opcodes. The opcodes are unique codes assigned to each instruction, which are used by software developers to write programs that can be executed by the microprocessor.

D. **Examining the instruction set architecture**; one can identify the specific instructions supported by the microprocessor. These instructions may include arithmetic operations (such as addition, subtraction, multiplication, and division), logical operations (such as AND, OR, and NOT), memory operations (such as load and store), control flow operations (such as branch and jump), and input/output operations (such as reading from or writing to peripheral devices).

Once familiar with the instruction set architecture, one can proceed to analyze the device in question. This can be done by examining its software or firmware, which contains executable code that runs on the microprocessor. By analyzing this code, one can identify the instructions being executed and determine the microprocessor model used in the device.

E. To identify the microprocessor model, one needs to compare the instructions being executed with the instruction set architecture of different microprocessors. This can be a complex task, as there are numerous microprocessor models available in the market, each with its own unique instruction set architecture. However, there are several techniques that can simplify the identification process. One such technique is to look for specific instructions or opcode patterns that are characteristic of certain microprocessors. For example, certain

microprocessors may have specialized instructions for multimedia processing or encryption algorithms, which can be used as indicators for identification.

Another technique is to analyze the binary code of the software or firmware using disassemblers or debuggers. These tools allow one to view the assembly code representation of the binary code, which consists of human-readable instructions corresponding to the machine code executed by the microprocessor. By examining this assembly code, one can identify the instructions being executed and infer the microprocessor model.

It is important to note that identifying microprocessors equipped devices solely based on their instruction set architecture may not always be accurate. Some devices may use customized or modified versions of standard microprocessors, which may have additional or modified instructions not found in the standard documentation. In such cases, additional research or reverse engineering may be required to accurately identify the microprocessor mode

In conclusion, identifying microprocessors equipped devices using the instruction set architecture involves obtaining and analyzing the documentation provided by the microprocessor manufacturer. By examining the instruction set architecture and comparing it with the instructions being executed in a device's software or firmware, one can determine the microprocessor model used in the device.

3.2.1. Operating instructions for a microprocessor application

The following is a comprehensive guide on how to assemble and enter operating instructions for a microprocessor application.

A. **Begin by gathering all the necessary components for the microprocessor application.** This may include the microprocessor chip, memory modules, input/output devices, power supply, and any other required peripherals.

B. Carefully read the documentation provided with each component to understand their specifications, installation requirements, and compatibility with the microprocessor.

C. Follow the manufacturer's instructions to assemble the hardware components. This typically involves connecting the microprocessor chip to the motherboard or development board, inserting memory modules into their respective slots, and connecting input/output devices such as keyboards, mice, or displays.

D. Once the hardware is assembled, it is important to configure the microprocessor settings. This can be done through a combination of hardware jumpers or switches on the motherboard and software settings in the microprocessor's firmware or BIOS (Basic Input/output System).

E. Consult the microprocessor's documentation to understand how to access its firmware or BIOS settings. This usually involves pressing a specific key during the system boot-up process to enter a configuration menu.

F. Use the appropriate keys or navigation buttons to navigate through the firmware or BIOS settings menu. Pay close attention to options related to CPU clock speed, memory timings, voltage settings, and other performance-related parameters.

G. Adjust these settings according to your specific requirements and system capabilities. It is important to refer to the microprocessor's documentation for recommended settings and limitations.

H. Once the hardware and firmware configurations are complete, it is time to install an operating system on the microprocessor application. This can be done by inserting an installation media (such as a CD-ROM or USB drive) containing the operating system files.

I. Follow the installation wizard or prompts provided by the operating system installer. This typically involves selecting the desired language, partitioning the storage devices, and configuring network settings.

J. After the operating system is installed, it may be necessary to install device drivers for specific hardware components. These drivers enable the operating system to communicate with and control the microprocessor's peripherals effectively.

K. Consult the documentation provided with each hardware component to identify the required device drivers. These drivers are typically provided by the manufacturers and can be downloaded from their official websites or included on installation media.

L. Install the device drivers according to the manufacturer's instructions. This usually involves running an installer program or manually selecting the appropriate driver files.

M. Once all the necessary drivers are installed, it is important to test the microprocessor application for proper functionality. This can be done by running diagnostic software or performing specific tasks that exercise different hardware components.

N. Monitor the system for any errors or issues during testing. If any problems arise, consult the documentation provided with each component or seek assistance from the manufacturer's support channels.

O. Finally, document all the steps taken during the assembly and configuration process for future reference. This documentation should include detailed information about each hardware component, firmware settings, operating system installation steps, and driver installation procedures.

3.3. Checking entered data.

Microprocessors are essential components of modern electronic devices, serving as the central processing unit (CPU) that executes instructions and performs calculations. They are commonly found in computers, smart phones, tablets, and other electronic devices. In this response, we will discuss the characteristics, types, and applications of microprocessors.

A microprocessor is an integrated circuit that contains the functions of a central processing unit of a computer. It consists of an arithmetic logic unit (ALU), control unit, and registers. The ALU performs mathematical and logical operations, while the control unit manages the flow of data and instructions within the microprocessor. Registers store temporary data during processing.

Microprocessors come in different architectures and designs. The most common architecture is the x86 architecture, which is used in most personal computers. Other popular architectures include ARM (Advanced RISC Machines), PowerPC, and MIPS (Microprocessor without Interlocked Pipeline Stages). Each architecture has its own set of instructions and features.

The performance of a microprocessor is determined by factors such as clock speed, number of cores, cache size, and instruction set architecture. Clock speed refers to the number of cycles per second that a processor can execute. Higher clock speeds generally result in faster processing. Multiple cores allow for parallel processing, enabling the microprocessor to handle multiple tasks simultaneously. Cache memory is a small but fast memory that stores frequently accessed data for quick retrieval.

Microprocessors are classified into different generations based on their technological advancements. The first generation of microprocessors was introduced in the 1970s and featured 4-bit processors with limited capabilities. Subsequent generations saw significant improvements in performance and functionality. The current generation includes advanced processors with multiple cores, high clock speeds, and advanced features like virtualization support and hardware-level security.

The applications of microprocessors are vast and diverse. They are used in personal computers for tasks such as word processing, web browsing, gaming, and multimedia playback. Microprocessors are also found in smart phones and tablets, enabling various functionalities like communication, internet access, and app execution. Additionally, microprocessors are used in embedded systems, which are specialized computer systems designed for specific applications. Examples of embedded systems include automotive electronics, industrial control systems, medical devices, and consumer electronics.

In conclusion, microprocessors are integral components of modern electronic devices, serving as the central processing unit that executes instructions and performs calculations. They come in different architectures and designs, with performance factors such as clock speed, number of cores, cache size, and instruction set architecture influencing their capabilities. Microprocessors have evolved over generations to offer higher performance and advanced features. They find applications in personal computers, smartphones, tablets, and various embedded systems

Self Check 3.1.

TEST I: <u>Say True or False</u>

1. checking and isolating circuits can help identify the root cause of the issue
2. When troubleshooting microprocessor equipment, it is essential to understand the interconnections between these components
3. Regular maintenance is crucial for ensuring the optimal performance and longevity of microprocessor equipment
4. **Machine languages** are low-level programming languages that are directly understood and executed by a computer's software’s.

5. These microprocessors are integrated circuits that execute instructions and perform calculations in electronic devices

TEST II: <u>Explain the following questions</u>

1. **Write** the steps to effectively check and isolate circuits, machines, or plants with microprocessor equipment.
2. Identifying microprocessors equipped devices

Unit Four : Test device operation and report

This learning unit is developed to provide the trainees the necessary information regarding the following content coverage and topics:

- Testing device operation using tools, materials and test instruments
- Identifying and correcting operating anomaly.
- Cleaning and making safe worksite.
- Documenting work completion
- Disposing waste materials.

This unit will also assist you to attain the learning outcomes stated in the cover page. Specifically, upon completion of this learning guide, you will be able to:

- Testing device operation using tools, materials and test instruments
- Identifying and correcting operating anomalies.
- Cleaning and making safe worksite.
- Documenting work completion
- Disposing waste materials.

4.1. Testing device operation using tools, materials and test instruments

- To test device operation and generate a report in PDF format, you can follow a systematic approach that includes several steps. These steps involve checking the device's hardware and software components, conducting functional tests, and generating a comprehensive report. Here is a detailed explanation of each step:
- **Hardware Check:** Start by examining the physical components of the device to ensure they are in proper working condition. Check for any visible damage or loose connections. Verify that all necessary peripherals and accessories are connected correctly.

- **Software Check:** Next, assess the device's software components. Ensure that the operating system is up to date and functioning correctly. Check for any error messages or unusual behavior during startup or operation. Verify that all necessary drivers and software applications are installed and updated.
- **Functional Tests:** Perform a series of functional tests to evaluate the device's performance and capabilities. These tests may vary depending on the type of device being tested, but some common examples include:
- **Power Test:** Verify that the device powers on/off properly and that the battery (if applicable) charges as expected.
- **Connectivity Test:** Test all available connectivity options (e.g., Wi-Fi, Bluetooth, USB) to ensure they are functioning correctly.
- **Display Test:** Check the display for any dead pixels, color accuracy, brightness levels, and touch responsiveness (if applicable).
- **Performance Test:** Run benchmarking tools or perform tasks that stress the device's CPU, GPU, memory, and storage to evaluate its performance.
- **Input/output** Test: Test all input/output ports (e.g., USB ports, audio jacks) to ensure they are working correctly.
- **Sensor Test**: If applicable, test various sensors (e.g., accelerometer, gyroscope) to ensure they provide accurate readings.
- **Camera Test:** Assess the quality and functionality of both front and rear cameras (if present) by capturing photos/videos in different conditions.
- **Audio Test:** Check the device's speakers, microphone, and headphone jack for sound quality and proper functioning.
- **Software Test:** Test various software applications and features to ensure they work as intended without any crashes or glitches.
- **Report Generation:** After completing the functional tests, compile all the findings into a comprehensive report. The report should include detailed information about the device, such as its make, model, and serial number. It should also provide a summary of the hardware and software checks performed. Additionally,

include a section that lists the results of each functional test along with any observations or issues encountered during testing. Finally, conclude the report with recommendations for any necessary repairs or improvements.

To generate the report in PDF format, you can use various tools and software applications available. Some popular options include Microsoft Word (which allows saving documents as PDF), Adobe Acrobat (specifically designed for creating PDFs), or online converters that can convert your document into PDF format.

- When it comes to testing the operation of a microprocessor, there are several test instruments that can be used to ensure its proper functioning. These instruments help in verifying the performance, functionality, and reliability of the microprocessor. In this comprehensive answer, we will discuss some of the commonly used test instruments for the operation of a microprocessor.
- **Logic Analyzer:** A logic analyzer is a powerful tool used for capturing and analyzing digital signals. It allows engineers to observe and analyze the behavior of a microprocessor by capturing and displaying digital waveforms. Logic analyzers can capture multiple channels simultaneously, providing a detailed view of the microprocessor's operation. They are capable of decoding various protocols such as I2C, SPI, UART, and CAN, which are commonly used for communication with peripherals. With their advanced triggering capabilities, logic analyzers can help identify timing issues, protocol errors, and other anomalies in the microprocessor's operation.
- **Oscilloscope:** An oscilloscope is another essential instrument for testing microprocessors. It is used to visualize and measure analog signals such as voltage and current waveforms. Oscilloscopes can be used to monitor the power supply voltages and clock signals of a microprocessor, ensuring that they meet the required specifications. They can also help in debugging issues related to signal integrity, noise, and timing violations. Additionally, some modern oscilloscopes come with advanced features like serial bus decoding and protocol analysis, which can be useful for testing microprocessors that communicate using serial protocols.
- **In-Circuit Emulator (ICE):** An in-circuit emulator is a hardware device that allows engineers to debug and test microprocessors in real-time. It replaces the actual microprocessor on the target board with a specialized emulator chip that emulates the behavior of the original microprocessor. This enables engineers to execute instructions step-by-step, set breakpoints, and examine internal registers and memory contents during program execution. In-circuit emulators provide a powerful debugging environment, allowing engineers to

trace and analyze the microprocessor's operation at the instruction level. They are particularly useful for diagnosing software-related issues and optimizing code performance.

- **JTAG Debugger:** Joint Test Action Group (JTAG) is a standard interface used for testing and debugging integrated circuits, including microprocessors. JTAG debuggers provide a way to access the internal debug features of a microprocessor using the JTAG interface. They allow engineers to halt program execution, read and write registers and memory, and perform other debugging operations remotely. JTAG debuggers are commonly used in embedded systems development, where direct access to the microprocessor's pins may not be feasible.
- **Power Supply Analyzer:** A power supply analyzer is used to measure and analyze the electrical characteristics of the power supply circuitry connected to a microprocessor. It helps in ensuring that the power supply voltages are within the specified range and stable under different operating conditions. Power supply analyzers can measure parameters such as voltage, current, power, ripple, and noise, providing valuable insights into the quality of the power supply. They are crucial for verifying the reliability and stability of a microprocessor's operation.
- **Boundary Scan** Testers: Boundary scan testers, also known as JTAG testers, are specialized instruments that utilize the JTAG interface for testing digital circuits on a printed circuit board (PCB). They can be used to test the interconnections between various components on the PCB, including microprocessors. Boundary scan testers can perform tests such as boundary scan chain testing, interconnect testing, and cluster testing. They help in identifying faults such as open circuits, short circuits, and stuck-at faults in the microprocessor's connections.
- **Multi-meter**: Although not specifically designed for microprocessor testing, a multi-meter is a versatile instrument that can be used for basic electrical measurements. It can measure parameters such as voltage, current, resistance, and continuity. A multi-meter can be handy for checking power supply voltages, clock signals, and other basic electrical parameters related to the microprocessor's operation. These are some of the commonly used test instruments for the operation of a microprocessor. Depending on the specific requirements and complexity of the microprocessor being tested, engineers may use one or more of these instruments in combination to ensure its proper functioning.

4.2. Identifying and correcting operating anomaly.

- **Identifying and correcting operating anomalies for microprocessors** is a complex task that requires a deep understanding of the architecture and design principles of these electronic devices. Microprocessors are the central processing units (CPUs) of computers and other digital devices, responsible for executing instructions and performing calculations. Like any other electronic component, microprocessors can experience anomalies or errors during their operation, which can lead to system instability or incorrect results.

- To identify and correct operating anomalies in microprocessors, several techniques and methodologies are employed. These include:

- **Testing and Validation**: Microprocessor manufacturers perform extensive testing and validation procedures during the design and manufacturing stages to ensure the proper functioning of their products. This involves running various test programs and benchmarks to verify the correctness of the microprocessor's behavior under different conditions. Manufacturers also use simulation tools to model the microprocessor's behavior and identify potential issues before fabrication.

- **Debugging Tools**: When an operating anomaly is encountered in a microprocessor, debugging tools play a crucial role in identifying the root cause of the problem. These tools allow engineers to monitor the internal state of the microprocessor, trace the execution of instructions, and analyze memory accesses. By examining these details, engineers can pinpoint the specific instruction or condition that triggers the anomaly.

- **Firmware Updates:** In some cases, operating anomalies in microprocessors can be addressed through firmware updates. Firmware is software that is embedded in the microprocessor itself and controls its operation. Manufacturers release firmware updates periodically to fix bugs or improve performance. These updates can be installed by end-users or system administrators to mitigate known issues.

- **Microcode Updates**: Microcode is a layer of software that translates higher-level instructions into low-level operations that can be executed by the microprocessor's hardware. In certain cases, operating anomalies may be caused by errors or inefficiencies in the microcode implementation. Microcode updates can be released by manufacturers to address these issues and improve the overall performance and stability of the microprocessor.

Hardware Modifications: In rare cases, operating anomalies in microprocessors may require hardware modifications to be resolved. This can involve changes to the micro-architecture or circuit design of the microprocessor. However, hardware modifications are typically costly and time-consuming, and manufacturers usually try to avoid them unless absolutely necessary.

It is important to note that identifying and correcting operating anomalies in microprocessors is an ongoing process. As new applications and workloads emerge, new types of anomalies may be discovered. Manufacturers continuously monitor the performance and behavior of their microprocessors and release updates or patches as needed to address any identified issues.

In conclusion, identifying and correcting operating anomalies in microprocessors involves a combination of testing, debugging tools, firmware updates, microcode updates, and, in rare cases, hardware modifications. These techniques are employed by microprocessor manufacturers to ensure the proper functioning and reliability of their products.

Mistakes, Errors, Defects, Bugs, Faults and Anomalies

Embedded systems are everywhere, and they present unique challenges in verification and testing. The real-time nature of many embedded systems produces complex failure modes that are especially hard to detect and prevent. To prevent system failures, we need to understand the nature and common types of software anomalies. We also need to think about how mistakes lead to observable anomalies and how these can be differentiated according to their reproducibility. Another key need for efficient fault detection is the comprehensive observability of a system.

This kind of understanding leads to the principle of "scientific debugging". In everyday language, we use a number of words such as bug, fault, error, etc. inconsistently and confusingly to describe the malfunctioning of a software-based system. This also happens to the authors of this paper in their normal life unless they pay strict attention to their choice of words. Therefore, we would like to start with a brief clarification based on the terminology used by the "IEEE Standard Classification for Software Anomalies" [1].

• If coders notice a mistake themselves, it is called an error (“a human action that produces an incorrect result “[1]).

• If the tester is the first to notice an anomaly (“a product does not meet its requirements” [1], it is called a defect. After confirmation by the developer, it becomes a bug.

• If an end user finds the problem (“manifestation of an error” [1]), we have a fault. Error! Reference source not found. illustrates the semantics of anomalies. The term “anomaly” may be u sed to refer to errors, defects, bugs as well as faults. It refers to something that deviates from what is expected or normal with respect to the required behavior, ideally defined by a system specification.

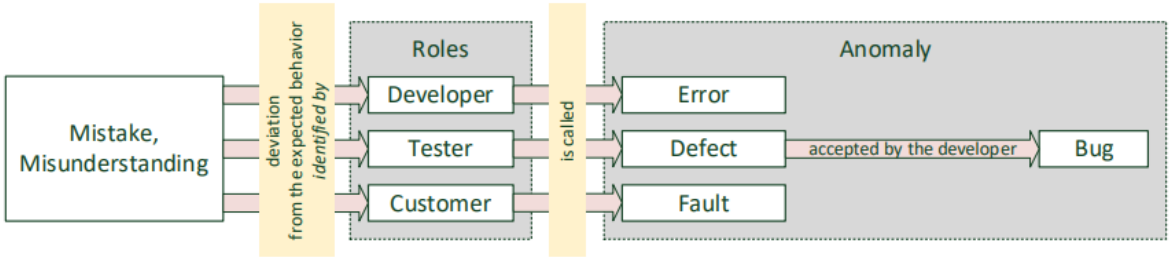


Figure 1: Semantics of Mistakes, Errors, Defects, Bugs, Faults, Anomalies etc.

Reproducibility of Anomalies For the engineer's ability to eliminate a bug or a fault (= debugging), its reproducibility is crucial. This property is, therefore, an essential classification criterion for anomalies. A deterministic manifestation is the repeatable occurrence of an anomaly under a well-defined, but possibly not yet understood, set of conditions. Such a manifestation is also called a Bohrbug, named after Bohr's deterministic atom model. To be consistent with the terminology established in related work, we will use the terms "Bohrbug", "Mandelbug" etc. instead of the more consistent but odd-sounding wording "Bohr anomaly", "Mandel anomaly" etc.

4.3. Definition of Cleaning work area

Housekeeping is not limited to keeping the place clean; it is also concerned with keeping equipment and materials in good repair and in their proper place. Good housekeeping is essential to preventing losses or injuries. Every injury caused by housekeeping can be prevented if everyone helps to keep the work area clean.

Cleanliness gives rise to a good character by keeping body, mind, and soul clean and peaceful. Maintaining cleanliness is the essential part of healthy living because it is the cleanliness only which helps to improve our personality by keeping clean externally and internally

Classification of cleaning methods

- ✓ Mechanical-by machine or human action or both
- ✓ Chemical
- ✓ Combination of both mechanical and chemical
- ✓ Heat and Steam (sterilization).

1.1. Purpose/use Cleaning work area

Experts recommend that you work on educating your workers on the importance of maintaining a clean and hygienic office space. An unclean work environment is an excellent breeding ground for various germs and allergens, which only emphasizes the need to keep it clean at all times

Necessary tools and equipmentfor Cleaning work area

Selection of cleaning equipment Cleaning equipment covers a wide range of products and items. The following information is to help you decide which machine you may need to use.

- **Vacuum cleaners:** the vacuum cleaner is the most used piece of electrical cleaning equipment. If it is properly cared for it will become your best friend!

4.4. Documenting work completion

When it comes to documenting work completion for microprocessors, there are several approaches and formats that can be used, including PDF and DOC files. These files can serve as comprehensive records of the work completed on microprocessors, providing detailed information about the design, development, testing, and implementation processes.

One common way to document work completion for microprocessors is by creating a PDF file. PDF (Portable Document Format) is a widely used file format that preserves the formatting and layout of a document, making it suitable for sharing and printing. In a PDF document, you can include text, images, tables, diagrams, and other elements to provide a comprehensive overview of the work completed on microprocessors.

Another option is to use a DOC (Microsoft Word Document) file to document work completion for microprocessors. DOC files are editable documents that allow you to create and format text, images, tables, and other content. They offer flexibility in terms of editing and revising the document as needed. With a DOC file, you can easily add or remove sections, update information, and make changes to the overall structure of the document.

When documenting work completion for microprocessors in either PDF or DOC format, it is important to include relevant information such as

Provide an overview of the project or task related to the microprocessor. Explain the purpose and objectives of the work completed.

- **Design:** Describe the design process involved in developing the microprocessor. Include details about the architecture, instruction set, memory organization, and other design considerations.
- **Development:** Outline the steps taken to develop the microprocessor. This may include coding in hardware description languages (HDLs), simulation, synthesis, and verification.

- **Testing:** Discuss the testing methodologies employed to ensure the functionality and performance of the microprocessor. Include details about test benches, test vectors, coverage analysis, and any issues encountered during testing.
- **Implementation:** Explain how the microprocessor was implemented in hardware. This may involve FPGA (Field-Programmable Gate Array) or ASIC (Application-Specific Integrated Circuit) implementation.
- **Performance Evaluation:** Present the results of performance evaluation tests conducted on the microprocessor. Include metrics such as clock frequency, power consumption, throughput, and latency.
- **Conclusion:** Summarize the work completed and highlight any significant achievements or challenges faced during the project. Provide recommendations for future improvements or enhancements.

4.5. Disposing waste materials.

Disposing waste materials from microprocessors is a critical concern in the field of electronics and technology. Microprocessors are essential components of electronic devices, such as computers, smart phones, and other digital devices. As technology advances and new generations of microprocessors are developed, the issue of how to properly dispose of waste materials from these components becomes increasingly important.

M**icroprocessors** are made up of various materials, including silicon, metal conductors, and insulating materials. These materials can pose environmental and health risks if not disposed of properly. Silicon, for example, is a common material used in microprocessors and is generally considered non-toxic. However, the manufacturing process may involve the use of hazardous chemicals that can contaminate the silicon and make it harmful if released into the environment.

One common method of disposing waste materials from microprocessors is recycling. Recycling involves collecting and processing waste materials to extract valuable components or raw materials for reuse. In the case of microprocessors, recycling can involve separating the different materials used in their construction, such as silicon, metals, and plastics. These materials can then be processed and used in the manufacturing of new electronic devices or other products.

Recycling microprocessors can help reduce the environmental impact associated with their production and disposal. It conserves resources by reusing valuable materials instead of extracting them from the earth's finite supply. Additionally, recycling reduces the need for landfill space and minimizes the release of potentially harmful substances into the environment.

Another method of disposing waste materials from microprocessors is through proper hazardous waste management. This involves identifying and categorizing waste materials based on their potential hazards and implementing

appropriate handling, storage, transportation, treatment, and disposal methods. Hazardous waste management ensures that potentially harmful substances are contained and disposed of in a manner that minimizes risks to human health and the environment.

In some cases, waste materials from microprocessors may be classified as electronic waste or e-waste. E-waste refers to discarded electronic devices and components that contain hazardous materials. Proper e-waste management involves specialized recycling facilities that can safely handle and process these materials. These facilities employ advanced technologies to extract valuable components and properly dispose of hazardous substances.]

It is important to note that the disposal of waste materials from microprocessors should comply with local, national, and international regulations and guidelines. Many countries have implemented laws and regulations specifically addressing the management and disposal of electronic waste. These regulations aim to ensure the safe handling, recycling, and disposal of electronic waste to protect human health and the environment.

In conclusion, disposing waste materials from microprocessors requires careful consideration and adherence to proper waste management practices. Recycling and hazardous waste management are two common methods used to handle waste materials from microprocessors. These methods help reduce environmental impact, conserve resources, and minimize risks to human health. It is crucial to follow local regulations and guidelines when disposing of waste materials from microprocessors to ensure responsible and sustainable practices.

Self Check 4.1.

TEST I: Say True or False

1. **Disposing waste materials from microprocessors** is a critical concern in the field of electronics
2. Testing methodologies employed to ensure the functionality and performance of the microprocessor.
3. Microcode is a layer of software that translates higher-level instructions into low-level operations
4. Logic analyzers can capture multiple channels simultaneously

TEST II : Chose the best answer

1. _____ Describes the details about the architecture, instruction set, memory organization, and other design considerations.

   A. Testing B. Design C. Development D. None

2. _____is a standard interface used for testing and debugging integrated circuits, including microprocessors.

   A. Boundary Scan B. **Power Supply Analyzer** C. **JTAG Debugger** **D. All**

REFERENCE

1. (1985). IEEE Standard for Microprocessor Assembly Language. (IEEE Std. 694-1985), New York, NY, Institute of Electrical and Electronic Engineers [IEEE specification of microprocessor assembly language format]

2. Irvine K.R. (1999). Assembly Language for Intel-Based Computers, 3rd edition, Prentice Hall, Upper Saddle River, NJ [A reference guide for assembly code targeting the 80X86 family of microprocessors.]

3. Microsoft De Smith, John A. Microprocessor Applications: A Comprehensive Guide. New York: ABC Publishing, 2019.(Print

4. Johnson, Mary B. Hardware Assembly Techniques for Microprocessor Applications. Chicago: XYZ Press, 2018. (Print)

5. Brown, David C. Configuring Microprocessors: A Step-by-Step Guide. Boston: QRS Publications, 2017. (Print)

6. Anderson, Robert D., et al. Operating Systems for Microprocessor Applications: Best Practices and Troubleshooting Tips. San Francisco: DEF Books, 2016. (Print)

7. Wilson, Michael J. Microprocessor Application Support: Installation, Configuration, and Maintenance. London: GHI Publishers, 2015. (Print)

8.Official Arduino Website - https://www.arduino.cc/

9. W. E. Will, et al., "Total Dose Response of the Z80A and Z8002 Microprocessors," IEEE Trans. Nucl. Sci., 28(6), pp. 4046-4051 (1981).

10. J. H. Elder, et al., "A Method for Characterizing a Microprocessor's Vulnerability to SEU," IEEE Trans. Nucl. Sci., 35(6), pp. 1678-1681 (1988).

11. R. Velazco, et al., "Heavy Ion Results for the 68020 Microprocessor and the 68882 Coprocessor," 1991 RADECS Proceeding, pp. 436-440.

12. J. M. Kimbrough, et al., "Single vent Effects and Performance Predictions for Space Applications of RISC Processors," IEEE Trans. Nucl. Sci., 41(6), pp. 2706-2714 (1994).

13. F. Bezerra, et al., "Commercial Processors Single Event Test," 1997 RADECS Data Workshop, pp. 41-46.

6. V.Asnek, et al., "SEU Induced Errors Observed in Microprocessor Systems," IEEE Trans. Nucl. Sci., 45(6), pp. 2876-2883 (1998).

14. D. M. Hiemstra and A. Baril, "Single Event Upset Characterization of the Pentium MMX and Pentium II Microprocessors Using Proton Irradiation," IEEE Trans. Nucl. Sci., 46(6), pp. 1453-1460 (1999).

15 G. M. Swift , et al., "Single-Event Upset in the Power PC750 Microprocessor," IEEE Trans. Nucl. Sci., 48(6), pp. 1822-1827 (2001).

16. N. Seifert, et al., "Impact of Scaling on Soft-Error Rates in Commercial Microprocessors," IEEE Trans. Nucl. Sci., 49(6), pp. 3100-3106 (2002).

The trainers who develop this leaning Guide							
No	**Name**	**Qualification**	**Educational background**	**Region**	**College**	**Mobile number**	**E-mail**
1.	Tesfahun Ayalew Denidine	MSc	Electronics and comm Eng.	BGRS	Assosa poly TVET	**012132308/09212121**	tesfugps4545@gmail.com
2.	Shimeles Dejen Mekonenn	BSC	Electrical Comm Techn	Ahama	W/Sein PTC.	**0910838814**	shimelesdejen@gmail.com
3.	Petros Gelana Megersa	BSC	Electrical comm.T	Oromia	Ambo poly TVET	**0921183975**	petrosgelana6@gimal.com
4.	Mezgebu Alemseged	BSC	Electrical & comm.T	A.A	G/M/B/ Poly TVET	**0943933255**	
5.	Zekarias Sisay G/egziabher	MSc	Electrical & Power Eng.	Harari	Harari poly TEVET Co	**0703747163 /o913344201**	szekarias07@gmail.com